A Mechatronic Haemodialysis System for the Treatment of Acute Renal Failure and Metabolic Disorders in Neonates

A thesis submitted to Middlesex University in partial fulfilment of the requirements for the degree of Doctor of Philosophy

N.Everdell
Engineering Systems Group
Middlesex University
Bounds Green Road
London
N11 2NQ

May 2001

Director of Studies: Dr. Y.B. Kavina

Supervisors: Prof. A.S.White, Dr. M.G.Coulthard and Dr. J.B.Lewis

CONFIDENTIAL

Abstract

The aim of this project was to produce a fully automated prototype system for the treatment of premature babies who are suffering from renal failure or metabolic disorders. These patients are difficult or impossible to treat conventionally, due to their very small total blood volume and their intolerance to donated blood. There was a strong case for developing a dialysis system specifically designed for the treatment of such patients.

The system is based on a manually operated device developed at the Royal Victoria Infirmary, Newcastle Upon Tyne. It differs from conventional dialysis methods in several ways. Blood access to the patient is via a single venous catheter. Only a very small amount of blood is needed to prime the extracorporeal circuit - this can be as little as 6.8 ml in the smallest patients. This compares very favourably with the volumes needed in conventional circuits, which are in the range of 15 - 40 ml. This small priming volume means that donated blood is not needed to prime the circuit. The clearance and ultrafiltration rates that can be achieved are independent of the rate that blood can be accessed from the patient, since the same blood passes back and forth through the haemofilter several times. The clearances that have been obtained experimentally are consistently above 40 % of the mean blood flow rate through the system. The largest mean blood flow rate available is 5 ml/min, so the maximum clearance is approximately 2 ml/min. The maximum ultrafiltration rate that can be obtained is 50 ml/h.

The new system is more effective at treating premature babies than conventional dialysis circuits. The hand driven system was tested *in vivo* and found to work well, so the automated system was developed on a solid foundation.
A prototype system has been successfully developed and tested. This thesis details both the development and the testing. The new system uses stepper motors and DC servo motors for actuation, and is controlled by Labwindows/CVI and NIDAQ software running on a standard PC platform. The interface between the PC and the machine is provided by a National Instruments data acquisition board. A comprehensive single fault analysis of the safety of the system was undertaken, including both software and hardware.

In vitro testing covered several areas of operation. The accuracy of the ultrafiltration process was established. The clearance rates that could be achieved were determined. The amount of damage caused to the blood by the system was

also tested. This was found to be well within acceptable clinical limits. *In vivo* testing established the feasibility of using a computer algorithm to control the withdrawal of blood from the patient. Finally, the system was successfully used to treat a patient with an in-born metabolic disorder.

In summary, a new system has been developed that is superior to any other treatment method currently available for neonates with these types of disorders.

Acknowledgements

Many people have given invaluable help throughout this project. Thanks go firstly to my director of studies, Dr. Y.B. Kavina, and to my other supervisors, Prof. A.S. White, Dr. M.G. Coulthard and Dr. J.B. Lewis. Peter Burns provided a great deal of help in the workshop. Fausto Renda worked on various aspects of the project. Thanks also go to Aboubaker Lasebae and Bernard Parsons for their help and advice. Many people at the Royal Victoria Infirmary and Newcastle University also gave valuable assistance: Val Dixon, Peter Lancaster, Mike Keir, Drs. Jim and Jen Cavet to name but a few. For their help with clinical matters, I'd also like to thank Dr. David Price and Dr. Ruth Ayling. Last but definitely not least, thanks go to Helena Drakakis, Graham Peacock, Paul Penfold, and John Shurety for all the help and support they've given.

Contents

Abstract i

Acknowledgements iii

Contents iv

List of Figures xi

List of Tables xiii

Glossary of Terms xiv

Chapter 1. Introduction

1.1 Background to the Present Study 1

1.2 Basic Anatomy and Physiology of the Kidney 2

1.3 Disorders of the Kidney 5

1.4 Dialysis Therapy 6

1.5 The History of Dialysis Therapy 8

1.6 Synopsis of the Present Study 10

Summary 11

Chapter 2. Literature Review

Introduction 20

2.1 Preterm Birth and Paediatric Renal Failure 20

2.2 The Treatment of Renal Failure 21

2.3 Sensors and Monitoring 24

2.4 Modelling and Analysis of Renal Replacement Therapy 25

2.5 Control Systems 33

2.6 Future Developments 34

2.7 Assessment of Manual System 35

2.8 Work of a Similar Nature 37

2.9 Miscellaneous Literature 38

Summary 39

Chapter 3. Specification of Prototype System

Introduction 43

3.1 Extracorporeal Apparatus 43

3.2 Mechanical Apparatus 43

3.3 Blood Withdrawal and Return 44

3.4 Ultrafiltration 44

3.5 User Interface and Control System 45

3.6 Safety Requirements 45

3.7 Other Requirements 46

Summary 46

Chapter 4. Initial Design Work

Introduction 47

4.1 Design of Extracorporeal Apparatus 47

4.2 Mechanical Actuation of Syringes 48

4.3 Control System 49

4.3.1 *Control Hardware* 49

4.3.2 *Control Software* 51

Summary 51

Chapter 5. Design of Mechanical and Electromechanical Components

Introduction 58

5.1 Syringe Actuation 58

5.1.1 *Filtration Phase Calculations* 58

5.1.2 *Blood Withdrawal and Return Phase Calculations* 59

5.1.3 *Motor, Gearbox and Lead Screw Selection* 63

5.1.4 *Linear Actuator Mechanism* 63

5.1.5 *Positional Feedback* 64

5.1.6 *Syringe Barrel Holders* 65

5.2 Three Way Tap Drivers 65

5.2.1 *Actuation* 65

5.2.2 *Detection* 66

5.2.3 *Connection between Driver and Tap* 67

5.2.4 *Initial Design of System* 67

5.2.5 *Three Way Tap Retainers* 67

Summary 67

Chapter 6. Design of the Interface Electronics

Introduction 75

6.1 Syringe Drivers 75

6.1.1 *Stepper Motor Drive Boards* 75

6.1.2 *Stepper Motor Series Resistors* 75

6.1.3 *5v - 12v Conversion Circuitry* 76

6.1.4 *Position Feedback Microswitch Circuits* 76

6.1.5 *Power Cut Off Microswitches* 77

6.2 Three Way Tap Drivers 77

6.2.1 *Motor Drive Circuit* 77

6.2.2 *Infrared Detectors* 78

6.3 Pressure Transducer Amplifier 79

6.4 DAQ Board Interface 81

6.5 Power Supply 81

Summary 83

Chapter 7. Construction, Testing and Design Modifications

Introduction 91

7.1 Construction and Testing of System Elements 91

7.2 Design Modifications 91

7.2.1 *Syringe Drivers* 91

7.2.2 *Three Way Tap Drivers* 92

7.2.3 *Pressure Sensing System* 95

7.2.4 *Syringe Barrel Holders* 96

Summary 96

Chapter 8. Software Development

Introduction 101

8.1 Data Acquisition Library Functions 102

8.2 Blood Filtration Phase - Control of Ultrafiltration Rate 103

8.3 Blood Withdrawal and Return 108

8.4 Three Way Tap Driver Control 113

8.5 Software Integration 116

8.6 User Interface and Callback Functions 118

8.7 System Initialisation 123

8.8 Error Processing 126

8.9 Miscellaneous Functions 127

8.10 Header Files 128

Summary 128

Chapter 9. System Safety

Introduction 131

9.1 Electrical Safety 131

9.2 Syringe Drivers 132

9.2.1 *Stepper Motors* 132

9.2.2 *Microswitches* 134

9.3 Three Way Tap Drivers 136

9.3.1 *Motor Drive Circuit* 137

9.3.2 *Infrared Transmitters* 137

9.3.3 *Infrared Detectors* 138

9.4 Pressure Transducer Amplifier 138

9.5 Computer System 139

9.6 Extracorporeal Circuit 140

9.7 System Initialisation 140

9.8 Safety in Normal Operation 140

Summary 141

Chapter 10. Testing the Prototype System

Introduction 142

10.1 Testing with Water 142

10.2 Testing with Blood 142

10.2.1 *Redesign of Three Way Tap Drivers* 142

10.2.2 *Blood Stirrer* 143

10.2.3 *Method* 143

10.2.4 *Results* 144

Summary 147

Chapter 11. Clinical Testing

Introduction 152

11.1 Addition of Dialysis to the System 152

11.2 Dialysis Testing 152

11.2.1 *Method* 154

11.2.2 *Results and Conclusions* 156

11.3 Haemolysis Testing 162

11.3.1 *Method* 162

11.3.2 *Results and Conclusions* 163

11.4 Blood Access Testing 165

11.5 Clinical Use of System 166

11.5.1 *Modifications to System* 166

11.5.2 *Treatment* 166

11.5.3 *Results and Conclusions* 167

Summary 171

Chapter 12. Conclusions and Proposals for Further Work

Introduction 184

12.1 Conclusions 184

12.2 Proposals for Further Work 187

Summary 190

13. References 191

14. Bibliography 201

Appendices

A. Final Construction of Prototype A.1

A.1 Construction of Electronic Circuitry A.1

A.2 Photodetector Mountings A.1

A.3 Final Assembly of Electronics A.1

A.4 Casing A.2

B. Example of Filtration Rate Calculation B.1

C. Complete Costing of Project C.1

D. Program Listings D.1

E. Operating Instructions for Haemodialysis System E.1

List of Figures

1.1 Original Circuit Configuration 12
1.2 Cross Section through an Adult Kidney 13
1.3 The Nephron 14
1.4 The Glomerulus 15
1.5 The Principle of Haemodialysis 16
1.6 The Countercurrent Mechanism 17
1.7 Abel's Apparatus 18
1.8 Kolff's Rotating Drum Dialyser 19

2.1 Solute Concentration vs. Distance along Haemofilter 39
2.2 A System to Recycle Dialysis Fluid 40
2.3 The Dialysis System designed by De Virgiliis 41
2.4 The Ariadne 1 Dialysis System 42

4.1 Tube Clamp Driven System 53
4.2 Final Configuration of Circuit 54
4.3 Horizontally Opposed Configuration 55
4.4 Two Way Circuit using One Syringe Pump 55
4.5 Two Way Circuit using Two Syringe Pumps 56
4.6 Maintenance of TMP in Secondary Cycle 57

5.1 Graph of Ultrafiltration Rate vs. Transmembrane Pressure 69
5.2 Linear Actuator Design 1 69
5.3 Linear Actuator Design 2 70
5.4 Linear Actuator Design 3 70
5.5 Syringe Driving System 71
5.6 Connecta 3 Way Tap 72
5.7 Photodetector Packaging 72
5.8 Tap Driver Photodetectors 73
5.9 Original 3 Way Tap Driver 73
5.10 Three Way Tap Retainer 74

6.1 Stepper Motor Wiring 84
6.2 5 V / 12 V Converter Circuit 84
6.3 Microswitch Interface Circuit 85
6.4 Motor Drive Circuit Design 1 85
6.5 Motor Drive Circuit Design 2 86
6.6 Motor Drive Circuit Design 3 86
6.7 Final Motor Drive Circuit Design 87
6.8 Infrared Transmitter and Detector 87
6.9 Pressure Transducer Differential Amplifier 88
6.10 DAQ Board Connections 89
6.11 LED Indicator Board 90
6.12 Schematic Diagram of Power Supply System 90

7.1 New 3 Way Tap Driver 98
7.2 Calculation of Potential Divider for Tap Driver Circuit 98
7.3 Modification to Pressure Transducer Amplifier 99
7.4 Syringe Barrel Holder 100

8.1 Graphical User Interface 129
8.2 Overall Structure of Control Software 130

10.1 Circuit Diagram of Blood Stirrer 149
10.2 Blood Stirrers 149
10.3 Graph of Error in Ultrafiltration Rate 151

11.1 Circuit for Dialysis Testing 172
11.2 Clearances for 10x Dialysate Flow Rate (10 ml syringe) 175
11.3 Clearances for 20x Dialysate Flow Rate (10 ml syringe) 175
11.4 Clearances for Low and High Dialysate Flow Rates (10 ml syringe) 176
11.5 Clearance vs. Filtration Time 176
11.6 Clearances for 10x Dialysate Flow Rate (25 ml syringe) 177
11.7 Clearances for 20x Dialysate Flow Rate (25 ml syringe) 177
11.8 Clearances for Low and High Dialysate Flow Rates (25 ml syringe) 178
11.9 Comparison between clearance and GFR 178
11.10 Comparison of Experimental Data with Models of Clearance 179
11.11 Haemolysis - 1st Test Results 180
11.12 Haemolysis - 2nd Test Results 181
11.13 Umbilical Access 182
11.14 Close Up of System in Operation 182
11.15 Overview of System in Operation 183

A.1 Matrix Board 1 A.3
A.2 Matrix Board 2 A.3
A.3 DAQ Board Interface Circuit A.4
A.4 DC - DC Converter Board A.4
A.5 Underside of Right Base Plate A.5
A.6 Underside of Left Base Plate A.5
A.7 Photodetector Mountings A.6
A.8 Eurocard Subrack A.6
A.9 Case A.7

E.1 Connecting the System Together E.5
E.2 Extracorporeal Circuit E.6
E.3 Graphical User Interface E.7

List of Tables

4.1	Blood Flow Control	52
4.2	Syringe Actuation	52
4.3	Syringe Driver Design	52
7.1	Design Modifications	96
10.1	Water Test Results	148
10.2	Analysis of Water Test Data	148
10.3	Blood Test Results	150
10.4	Analysis of Blood Test Data	150
11.1	Test Parameters	153
11.2	Dialysis Test Results for 10 ml Syringe	173
11.3	Dialysis Test Results for 25 ml Syringe	174
11.4	Haemolysis Test Results for 10 ml Syringe	180
11.5	Haemolysis Test Results for 25 ml Syringe	181

Glossary of Terms

Arteriovenous Haemofiltration A method of haemofiltration whereby the blood is withdrawn from the patient via an artery, and returned via a vein.

Cannula Another name used for a catheter, i.e. a tube used to obtain access to a vein or an artery through the skin.

Catheter A tube that can be inserted into a body cavity (e.g. a blood vessel) so that substances can be administered and removed from the body.

Clearance A measure of the efficiency with which the kidneys (or a dialysis machine) remove substances from the blood. It is defined as the volume of blood that is completely cleared of a given solute in unit time.

Countercurrent Mechanism A mechanism used to maximise the clearance obtained from a dialysis machine. The blood and the dialysate fluid flow in opposite directions in the haemofilter. This increases the average concentration gradient across the filter and so increases the diffusion of substances across the semipermeable membrane.

Creatinine An organic molecule that is a product of muscle metabolism and is excreted by the kidneys.

Dialysate The fluid that is used to exchange substances with the blood inside the haemofilter. This usually consists of purified (but not sterile) water with various salts dissolved in it.

Erythropoietin A hormone produced by the kidneys that stimulates the manufacture of red blood cells in the bone marrow.

Extracorporeal Outside the body.

Frusemide A powerful diuretic drug that increases urine output and reduces total blood volume, thereby reducing blood pressure in renal failure.

Haematocrit The percentage by volume of the blood that consists of red ? blood cells.

Glomerulus A spherical structure situated at the beginning of each kidney tubule. The first stage of blood filtration occurs in this structure, through the mass of capillaries contained within it.

Haemodialysis The process of removal of plasma solutes by diffusion down their concentration gradients across a semipermeable membrane. The process does not depend on the bulk movement of water.

Haemofilter The device that performs the exchange of substances between the blood and the dialysate fluid. Modern haemofilters consist of many small semipermeable tubes packed together inside a larger plastic tube. The blood flows inside the small tubes and the dialysate circulates outside the tubes.

Haemofiltration The process of removal of plasma solutes by bulk convection of protein free plasma water across a semipermeable membrane.

Haemoglobin A substance contained in red blood cells that provides the main oxygen carrying capability of the blood.

Haemolysis The escape of haemoglobin into the plasma, caused by the breakdown of the cell membrane of the red blood cells.

Heparin An anticoagulant drug - used to prevent clotting of the blood.

Hirudin The first anticoagulant drug - derived from leeches.

Hypotension Low blood pressure.

Hypoxia A condition of low levels of oxygen in the tissues.

Infusion Pump A device used to control the delivery of intravenous drugs to a patient. It often takes the form of a motor driven syringe.

In Vitro Outside the living body.

In Vivo Inside the living body.

Luer Lock A standardized connection system in widespread use in clinical equipment. It allows a wide range of equipment (e.g. syringes, tubing) to be easily connected together.

Lumen A general term used to refer to the space inside an artery, vein or tube.

Necrotising Enterocolitis A disease of the gut common in preterm infants - loss of blood supply to the gut leads to necrosis. Surgery is often needed to remove the necrotic sections of the gut.

Neonate Newborn baby.

Peritoneal Dialysis A method of dialysis that uses the peritoneum (the membrane surrounding the gut) as the semipermeable membrane. Dialysis fluid is infused into the peritoneal space through the abdominal wall. Solutes from the blood pass through the peritoneum into the fluid, which is subsequently drained from the peritoneal space, thus removing the unwanted solutes from the body.

Plasma The fluid portion of the blood in which the formed elements (the cells) are suspended.

Recirculation Fraction In the extracorporeal circuit of a dialysis machine, not all the blood is returned to the body at the end of each cycle. Some will remain in the access lines. This blood is 'recirculated' to the machine, resulting in a loss of efficiency, since the same blood is effectively dialysed more than once. The recirculation fraction is the volume of this recirculating blood expressed as a percentage of the total extracorporeal volume.

Respiratory Distress Syndrome A common condition of preterm babies - insufficient surfactant is produced by the lungs, leading to a decreased respiratory capability.

Right Atrium The chamber of the heart that receives deoxygenated venous blood from the vena cava.

Semipermeable Membrane A membrane that allows the passage of certain substances but restricts the passage of others.

Subclavian Vein A large vein that lies underneath the clavicle (collar bone). It is often used as a point of entry for a venous catheter.

Surfactant A chemical produced in the lining of the lungs, which reduces surface tension and makes lung tissue expand more readily.

Transmembrane Pressure The pressure difference that exists across a semipermeable membrane, which drives the passage of substances across the membrane.

Ultrafiltration A more general term than haemofiltration - it covers filtration both by the use of a blood circuit and by use of the peritoneum as a semipermeable membrane.

Vena Cava A large vein that opens into the right atrium of the heart, through which most of the deoxygenated blood returning from the body passes.

Venovenous Haemofiltration A method of haemofiltration by which blood is withdrawn from the patient via a vein and also returned via a vein.

CHAPTER 1. INTRODUCTION

1.1 Background to the Present Study

In recent years there has been a steady improvement in the survival rates of premature babies. This is not due to any one major advance in this field of medicine. Rather it is because of steady progress in a number of areas of neonatal intensive care. The equipment available has improved as well as the actual methods of treatment.

However, these general improvements have exposed one area of weakness, that of the management of acute renal failure (ARF). Up to 8 % of neonates admitted to intensive care can be expected to develop renal failure [1,2]. In most of these cases the renal disease is secondary to some other serious condition. Common amongst these are respiratory distress syndrome (RDS), necrotising enterocolitis and cardiac disease. RDS is the most common complication of preterm birth. It is caused by insufficient production of surfactant in the immature lungs - this prevents proper lung function.

For higher birthweight babies, a range of dialysis treatments are available. It is generally accepted [3] that peritoneal dialysis is the preferred treatment. If this is not feasible (e.g. because of abdominal surgery) then either arteriovenous or venovenous extracorporeal circuits can be used.

However, for babies with birthweights below 1 kg, the treatment options available are much more limited. Necrotising enterocolitis is a common complaint in premature babies - abdominal surgery is usually needed to correct this condition, and this makes peritoneal dialysis very difficult, due to the increased risk of abdominal infection (peritonitis). Continuous arteriovenous filtration requires a minimum mean arterial pressure of 40 - 50 mm Hg [4] to achieve a sufficient extracorporeal blood flow rate. A typical blood pressure for a very low birthweight baby is 45/30 mm Hg, so this criterion is not met. In theory this limitation could be overcome by using venous access and an externally pumped circuit - in practice it is difficult to achieve adequate blood flow with this technique as well.

If the problems of blood access can be overcome there still remains the difficulty of priming the circuit. It must be remembered just how small the total blood volume is in a 1 kg baby. A typical value is 40 ml [5] as compared with 5 litres in an average adult , i.e. just 0.8 % of the adult volume. This means that priming even

the smallest extracorporeal circuit with donated blood can produce massive, rapid changes in overall blood biochemistry. In very small babies, normal whole blood can never be used to prime the circuit. It contains large amounts of glucose (this keeps the cells alive) and all the calcium is removed from it. The processing it undergoes for preservation renders it very dangerous for babies of this size.

These difficulties led to the development of a manual dialysis system by the Royal Victoria Infirmary in Newcastle [6]. This is shown in figure 1.1. It's novel design overcomes both the problem of vascular access and of blood priming. It is different in principal from conventional dialysis equipment because it does not rely on continuous vascular access. Instead, it allows the removal of a fixed quantity of blood from the patient, followed by the repeated passing of this blood back and forth through a haemofilter until the desired amount of haemodialysis and/or ultrafiltration has been achieved. The blood is then returned to the baby. This process is repeated until biochemical control has been achieved.

The total extracorporeal volume of the circuit is very small (as little as 8.5 ml) so priming with donated blood is unnecessary.

The system was tested clinically and the results were very promising. It was much easier to achieve control of blood biochemistry than with conventional dialysis circuits. A typical patient might have a 3 hour session of ultrafiltration every 12 hours. Ideally these treatment times should be extended so that swings in blood biochemistry are minimised. This is an important factor in very low birthweight babies - they are very intolerant of sudden changes.

However, this system is not a practical one as it requires the constant attendance of a nurse throughout the whole treatment process. This is in addition to the intensive care nurse who is already treating the baby on a one to one basis. This project arose out of the need to automate this manual system and produce a practical clinical device that could provide both haemofiltration and haemodialysis.

1.2 Basic Anatomy and Physiology of the Kidney

This and subsequent sections of this chapter provide some basic background knowledge of the subject, which is necessary for the understanding of subsequent chapters.

The kidneys have several different functions:

1. The main one is to help maintain the body's internal environment at a relatively constant state, in the face of a constantly changing external environment. This is known as homeostasis. This is done mostly by regulating the volume and composition of the blood. Waste products of metabolism are excreted in the form of urine to maintain the composition, and excess water is removed to maintain the volume.

2. They contribute to the regulation of blood pressure by secreting the enzyme renin.

3. Synthesis of glucose during periods of fasting.

4. Participation in the synthesis of vitamin D, which is important in maintaining bone density.

5. Secretion of erythropoietin, a hormone that stimulates the production of red blood cells in the bone marrow. This is why patients with renal failure become anaemic and need regular injections of erythropoietin.

Overall, the main function of the kidneys is best described as one of regulation of the internal environment, rather than just the function of excretion.

A cross section through an adult kidney is shown in figure 1.2. The renal capsule forms a protective layer around the kidney. The organ itself is divided into 3 main regions. Outermost is the cortex, the middle region is the medulla, and the innermost is the pelvis. Together the cortex and the medulla constitute the parenchyma, which is the functional part of the kidney. The pelvis is the central cavity that collects the urine produced by the outer portions of the kidney. It projects into the parenchyma by means of extensions called calyces. These terminate in the papillae, through which flows the urine produced in the parenchyma. The pelvis empties into the ureter, which is the vessel that takes the urine from the kidney to the bladder. The blood vessels that supply the kidney are not shown in the figure. These are the renal artery and the renal vein.

The parenchyma of each kidney contains approximately 1 million nephrons, shown in figure 1.3. This is the functional unit of the kidney. It has two parts, the glomerulus and the renal tubule. The glomerulus is shown in more detail in figure 1.4. A cluster of capillaries is supplied and drained of blood by two arterioles (small

blood vessels). It is through this cluster of capillaries that the first stage of filtration occurs. A fluid that is essentially of the same composition as blood plasma passes from the capillaries into the Bowman's space outside. The Bowman's capsule surrounds the capillary cluster and directs the filtrate into the renal tubule. It is interesting to note the rate of production of this glomerular filtrate. Both kidneys together produce approximately 50 gallons in a 24 hour period. Obviously the vast majority of this filtrate must be reabsorbed to maintain fluid balance, and this is the function of the renal tubule.

The tubule itself loops down into the medulla and back up into the cortex before joining a larger collecting duct which descends back into the medulla. In this way a much longer tubule can fit into a smaller space. As the filtrate passes through the tubule, water, salts, proteins and other substances are reabsorbed by the blood vessels that supply the tubule. These same blood vessels also secrete certain other substances into the filtrate. As the vast majority of the water in the filtrate is reabsorbed, the final quantity of urine produced is much less than that of the glomerular filtrate, typically 1.5 to 2.5 litres per day for a normal adult. The urine flows from the collecting ducts into the renal pelvis and leaves the kidney through the ureter.

The concept of clearance is important in the evaluation of kidney function and also when assessing the efficiency of a dialysis system. Clearance is defined as the volume of blood plasma (see glossary) that is completely cleared of a given solute per unit time. So it is a measure of the rate at which the kidneys can remove substances from the blood. Creatinine is often used to measure clearance. It is an organic molecule produced as a byproduct of muscle metabolism. The patient's urine is collected and measured over a period of 24 hours, so that the urine flow rate V (in ml/min) can be calculated. The creatinine concentration in the plasma P (in mg/ml) and in the urine, U (also in mg/ml) are determined. Since the rate of removal of creatinine from the plasma must be the same as its rate of removal from the body through the urine, a simple formula gives the plasma clearance for creatinine:

$$C = \frac{UV}{P} \qquad (1.1)$$

In summary, the kidney is a very complex organ, with many different functions that are vital to maintaining the homeostasis of the body. Many of the

physiological processes that occur are still not well understood.

1.3 Disorders of the Kidney

Disorders of kidney function can be divided into two broad categories - acute renal failure and chronic renal failure.

Acute renal failure [7] is characterised by the sudden deterioration of renal function and reduction in urine output. It is often reversible if prompt treatment is given. It is usually the result of some other systemic disease or trauma rather than a disorder of the kidney itself. A common cause is insufficient perfusion (blood flow) of the kidneys, which can have a variety of causes (e.g. loss of blood through haemorrhage resulting in low blood pressure). In premature babies, acute renal failure is the type of disorder most commonly seen.

The other category is chronic renal failure [7]. This is a more gradual loss of renal function, and is more often irreversible. It is usually the result of a progressive disease of the kidney itself. It is interesting to note the amount of redundancy that exists in kidney function. Only when more than 75 % of the functioning nephrons are lost through disease do any symptoms of illness appear. Up to this point the remaining nephrons enlarge and take over the work of those that have been lost. Once more than 75 % of nephrons are lost, then the levels of creatinine and urea in the blood will start to rise above normal, and the kidneys lose their ability to dilute and concentrate the urine. Only when 90% of the nephrons are lost is the patient said to be in end stage renal failure.

Some of the more common kidney conditions are described below.

Glomerulonephritis (Bright's Disease)

This is an inflammatory disease of the glomerulus. It is often caused by an infection elsewhere in the body (e.g. the throat). The immune response triggered by this infection starts to attack the glomeruli, making them inflamed and swollen. The membrane that filters the blood becomes more permeable and allows red blood cells and proteins to enter the glomerular filtrate. The damage to the glomeruli may become permanent.

Pyelonephritis

An inflammation of one or both kidneys that involves both the nephrons and the renal pelvis. Again, this is usually caused by an infection elsewhere in the body.

Polycystic Disease

The most common genetic disorder of the kidneys. The nephrons deform and dilate, forming fluid filled cysts. These cysts gradually increase in size and number, interfering with the normal kidney tissue. The kidneys themselves enlarge greatly, up to 60 times their normal size. The disease is progressive, but can be slowed by certain drugs and regulation of diet and fluid intake.

1.4 Dialysis Therapy

Both acute and chronic renal failure can be treated using dialysis[7]. In acute failure dialysis can be used as a temporary measure while the kidneys recover their normal function. The chronic kidney patient however requires long term dialysis until such time as a kidney transplantation can be performed.

There are two main types of dialysis, haemodialysis and peritoneal dialysis. They employ the same basic principle, although they use very different methods to put this principle into practice. The principle is that two fluids are separated by a semipermeable membrane. This membrane allows the passage of small organic molecules and salts between the two fluids, but prevents the passage of large molecules, such as proteins. In this way nitrogenous wastes can be removed from the blood without the loss of the proteins and formed elements (cells). In this way the dialysis machine can replace the excretory function of the kidneys.

Peritoneal dialysis (often known as continuous ambulatory peritoneal dialysis or CAPD) uses the peritoneum as the semipermeable membrane. The peritoneum is the membrane that lines the abdominal cavity and the outside of the gastrointestinal tract. A sterile mixture of salts dissolved in water is prepared, which is known as a dialysis fluid or dialysate. This is then introduced into the peritoneal cavity through a catheter inserted into the abdominal wall. This is left for several hours to allow the exchange of substances between the blood and the fluid to take place. The dialysate is then drained off. Peritoneal dialysis has the big advantage of being simple to perform. It's big disadvantage is that there is an increased risk of infections of the peritoneum (peritonitis). However, these can usually be controlled with antibiotics.

Haemodialysis uses an artificial semipermeable membrane situated outside the body. Figure 1.5 illustrates the principle of haemodialysis. Blood is withdrawn from an artery, usually in the arm, and is pumped through a haemofilter which provides the semipermeable membrane. The blood path through the haemofilter consists of many small hollow tubes made of a semipermeable polymer. The blood flows inside the tubes. The return path takes the blood back into the body through a vein in the arm. The hollow tubes are surrounded by dialysis fluid, which is pumped through the haemofilter in the opposite direction to the flow of blood. The hollow tube construction allows a large membrane surface area to be packed into a relatively small volume. As well as removing nitrogenous wastes, haemodialysis can also control the water content of the blood, which is equally important. Pumps can be used to create a pressure gradient between the blood and dialysate compartments in the haemofilter, which causes a net movement of water across the membrane from the blood to the dialysate. This is known as ultrafiltration.

As was mentioned above, the dialysis fluid and the blood are pumped through the haemofilter in opposite directions. This provides an important increase in the efficiency of the haemofilter. It is known as the countercurrent mechanism. Figure 1.6 illustrates the principle. The left hand diagram shows an example situation when the blood and dialysate flow in the same direction. A solute in the blood enters the filter at a concentration of 100 mmol/l. Diffusion across the semipermeable membrane occurs and at the end of the haemofilter the concentration in the blood has dropped to 20 mmol/l. At the same time the concentration of the same solute in the dialysate has risen from 0 to 20 mmol/l. So although there is a large concentration gradient across the membrane where the blood enters the filter, towards the end of the blood pathway the gradient has dropped to zero and no diffusion will occur here. However, if the dialysate flow direction is reversed (right hand diagram) the average concentration gradient will rise. Now, where the blood concentration has dropped to 20 mmol/l, the dialysate concentration is 0 mmol/l as the dialysate has just entered the filter. Therefore diffusion will still occur in this region of the filter.
In this way a favourable concentration gradient can be maintained throughout the entire length of the haemofilter.

The rate at which a haemodialysis system removes solute from the blood can be quantified using the concept of clearance in much the same way as is done with the kidney. Either the blood circuit or the dialysate circuit can be used. The concentration of the chosen solute is measured at the inlet to and the outlet from

the haemofilter. The flow rate through the filter is also measured. The clearance of the given solute can then be calculated using a formula similar to that used for the kidney. If the dialysate is used for the calculation, it is also necessary to measure the solute concentration in the blood plasma, to arrive at a final figure for blood plasma clearance. For a system used to treat an adult patient, a typical clearance value would be 200 ml/min. The clearances needed for the treatment of a neonatal patient are very much less, typically 1 or 2 ml/min.

1.5 The History of Dialysis Therapy

It is generally agreed [7, 8] that the foundations of the science of dialysis were laid by Thomas Graham, a 19th century Scottish chemist. He first described the differential movement of substances across a semipermeable membrane in 1854. His membranes were prepared from ox bladder. The first dialysis of human blood (in vitro) was performed by Richardson in 1889. He categorised substances in the blood into two groups: 'crystalloid' substances, which would pass through a membrane, and 'colloidal' substances, which would not.

The first in vivo experiments were described by Abel et al. in 1913 [9]. They constructed a vivi-diffusion apparatus from glass and celloidin tubing that was very similar in design to today's hollow fibre dialysers (figure 1.7). Each celloidin tube was 8 mm in diameter and 400 mm long. The apparatus consisted of 32 such pieces of tubing. They demonstrated that dialysis could be safely performed on dogs.

For experimenters at this time, there were two major practical problems - membranes and anticoagulation. Celloidin was a difficult substance to prepare and use, but was the only practical material available. The description of its preparation [10] reveals just how difficult it was to prepare and work with. It is derived from the explosive nitrocellulose (gun cotton). Since it was not commercially available, each researcher had to make their own membranes. On top of this, the finished product was very fragile and prone to cracking. Only when cellophane (originally used as a sausage casing) became commercially available in the 1920s were these problems solved.

The other major problem was anticoagulation. Whenever blood comes into contact with a foreign surface, it is liable to clotting. This reduces the efficiency of the dialyser and also presents a serious risk to the patient if a clot finds its way back into the body. Abel and others used hirudin as an anticoagulant [9, 11]. This was

isolated from the saliva of the European leech, found in the swamps of Yugoslavia. It was very expensive, and had toxic side effects. As well as this, the supply of leeches was curtailed by the outbreak of the first world war. There was no suitable alternative for many years, until the introduction of heparin, which is still in use today. Although it was first isolated in 1916, heparin was not available in a useful clinical form until 1935.

The first human dialysis was attempted in 1924 by Georg Haas, working in Germany. Although the procedure produced no ill effects in the patient, the low amount of clearance of toxic substances achieved meant that there was no therapeutic benefit. The first successful dialysis was performed by Kolff in 1945. He had developed his techniques in occupied Holland during the second world war. His apparatus was technically complex, and included a rotating drum that carried the blood filled cellophane tubing (figure 1.8). The drum rotated in a bath of dialysing fluid. The idea of this was to increase the diffusion gradient across the membrane, thereby increasing the clearance that could be achieved. After the war Kolff moved to the U.S.A. and there developed the Kolff - Brigham Artificial Kidney, which was the first commercial system. This was used to treat wounded soldiers in the Korean war [12] under mobile hospital conditions. This represented a major breakthrough, and led to a much greater acceptance of the idea of artificial kidney systems in the medical community. It is worth noting [13] that the medical profession was very slow to take up the idea of dialysis. It was viewed with suspicion by many leading medical professionals for much of the 1940s and 50s.

The next few years saw a number of new designs [14]. Some were refinements of the rotating drum concept, such as the machine produced by the Allis Chalmers company. Over 20 of these machines were produced before the project was abandoned due to lack of demand. This machine was still considered to be too complex and costly for routine clinical use. Westinghouse attempted to produce a more compact machine, employing a vertical rotating drum instead of the usual horizontal one. 3 of these machines were produced. It was not particularly successful for 2 reasons. It took too long to set up and it was of limited efficiency because of the small membrane surface area.

Later designs still used membranes in the form of coiled tubing, but these no longer rotated in the dialysis bath.

Still other designs arranged the membrane as a series of parallel plates stacked on top of each other.

In 1960, a patient with chronic renal failure was maintained on long term

dialysis for the first time. 1966 saw the introduction of the hollow fibre dialyser, the same basic design that is in use today.

The first recorded dialysis of a child dates from 1955 [15]. Of the five patients described, the youngest two were 7 years old. The Westinghouse coil machine was used. The problems of using equipment designed for adults with children were apparent. However, biochemical control was achieved and the principle of treating children was established. The lack of availability of suitable paediatric equipment continued until the late 1960s [16]. The resolution of this and other problems led to a greatly increased success rate for treatment of paediatric renal failure [17].

Since the early 1970s progress has been fairly rapid. Better membranes such as polysulphone have been developed. These materials are more biocompatible and allow much greater clearance rate for a given surface area, allowing dialysers to be much more compact than they were previously.
The other major area of development has been in the control of ultrafiltration. Early machines were unable to control the amount of fluid that was removed by convection. Advances in computer technology and control methods allow modern machines to carefully control fluid removal, greatly increasing patient tolerance to dialysis.

1.6 Synopsis of the Present Study

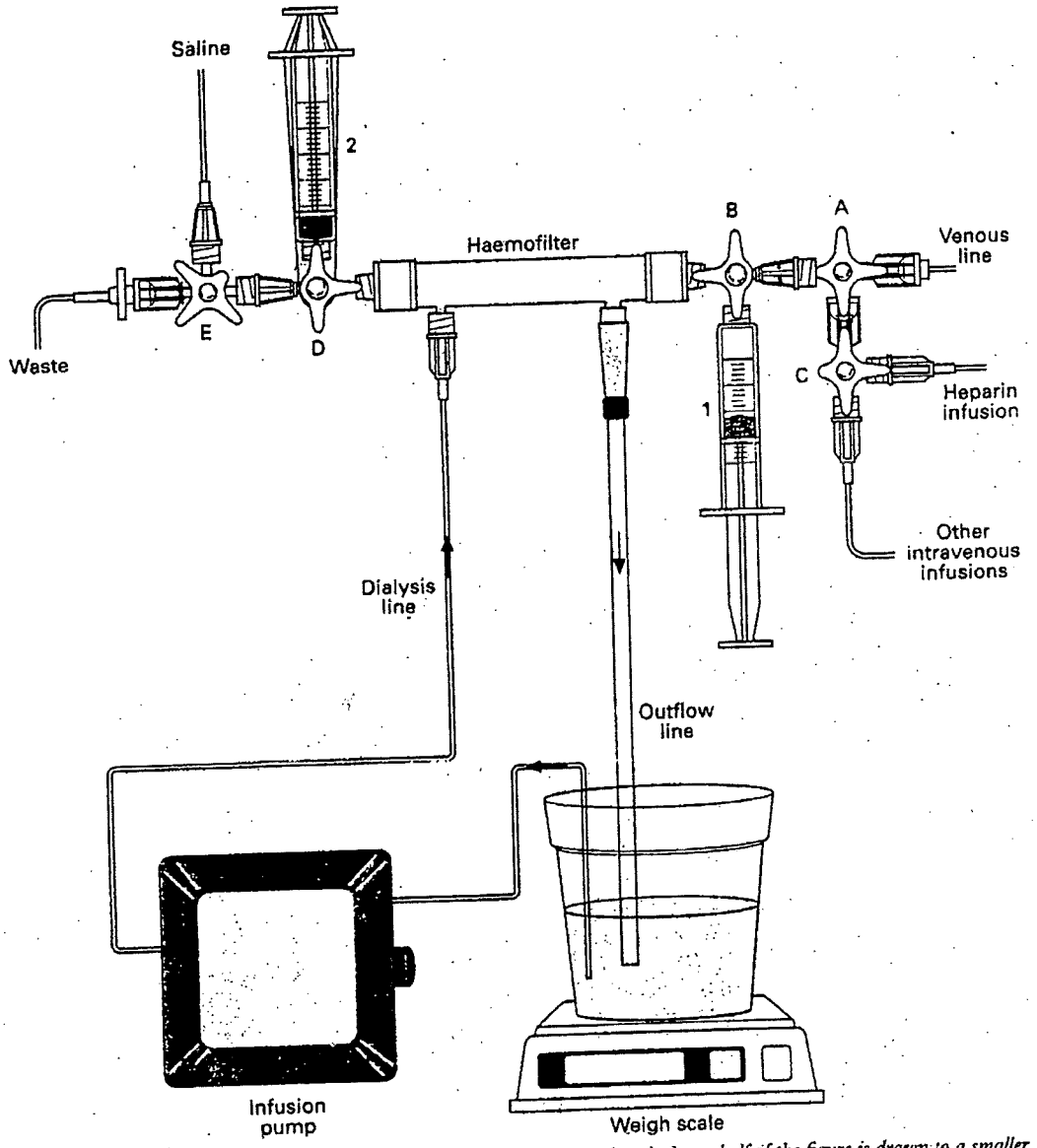
This chapter has provided the background knowledge necessary for the understanding of the project. The anatomy and physiology of the kidney was dealt with, as well as the common disorders of renal function. The current practice of dialysis therapy was then reviewed, followed by a history of the development of dialysis. Chapter 2 reviews the relevant literature that is available. An extensive literature search was undertaken to establish the unique nature of the current project. No other system using the same basic principles was discovered. Two systems bearing superficial resemblance were documented. These are reviewed in section 2.8. Chapter 3 deals with the specification of the prototype system that was the starting point of the system design. This was drawn up at the start of the project in consultation with the Royal Victoria Infirmary, Newcastle. Chapter 4 presents the initial design work that was done to lay the foundation of the system design. Chapter 5 concerns the detailed design of the mechanical and electromechanical elements of the system. The interface electronics are discussed in chapter 6. These provide the connection between the various sensors and

transducers and the data acquisition board that provides the interface to the PC. Chapter 7 deals with the construction and testing of the prototype, as well as the design modifications that were necessary. Chapter 8 addresses the development of the software used to control the system. This was a major part of the project and this is therefore a long chapter. Chapter 9 discusses system safety. Chapter 10 deals with the early testing of the system. This testing established the basic functioning of the system and led to various design modifications. Chapter 11 details the extensive clinical testing that was done to establish the operating characteristics of the system. It also includes a section on the first clinical use of the system. Finally, chapter 12 presents the conclusions that were reached and proposals for further work.

Summary

This chapter has described the clinical need that was the origin of the current project. The background knowledge of the subject that is required has been given. This includes the anatomy and physiology of the kidney, as well as the more common disorders of renal function. Current practise in dialysis therapy and the history of dialysis techniques have been reviewed.

At the start of the project, an extensive literature review was undertaken. This is presented in the next chapter.



Ultrafiltration and haemodialysis circuit for a very small infant. Note that the lower half of the figure is drawn to a smaller scale than the upper half.

Figure 1.1 Original Circuit Configuration [6]

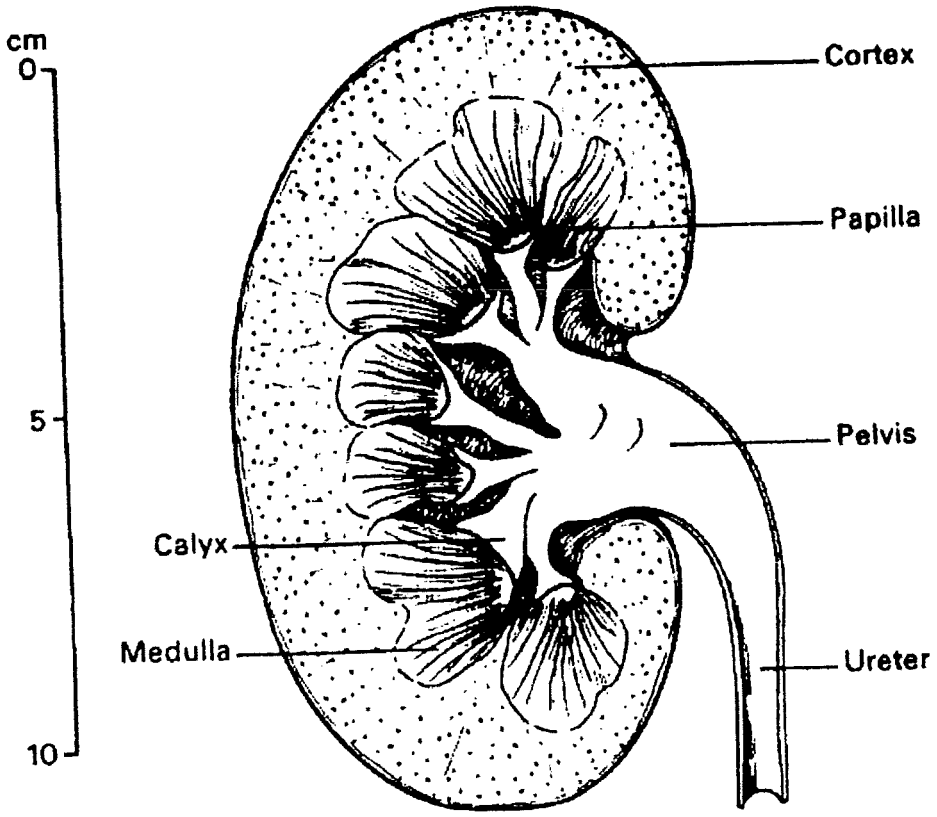


Figure 1.2 Cross Section through an Adult Kidney [19]

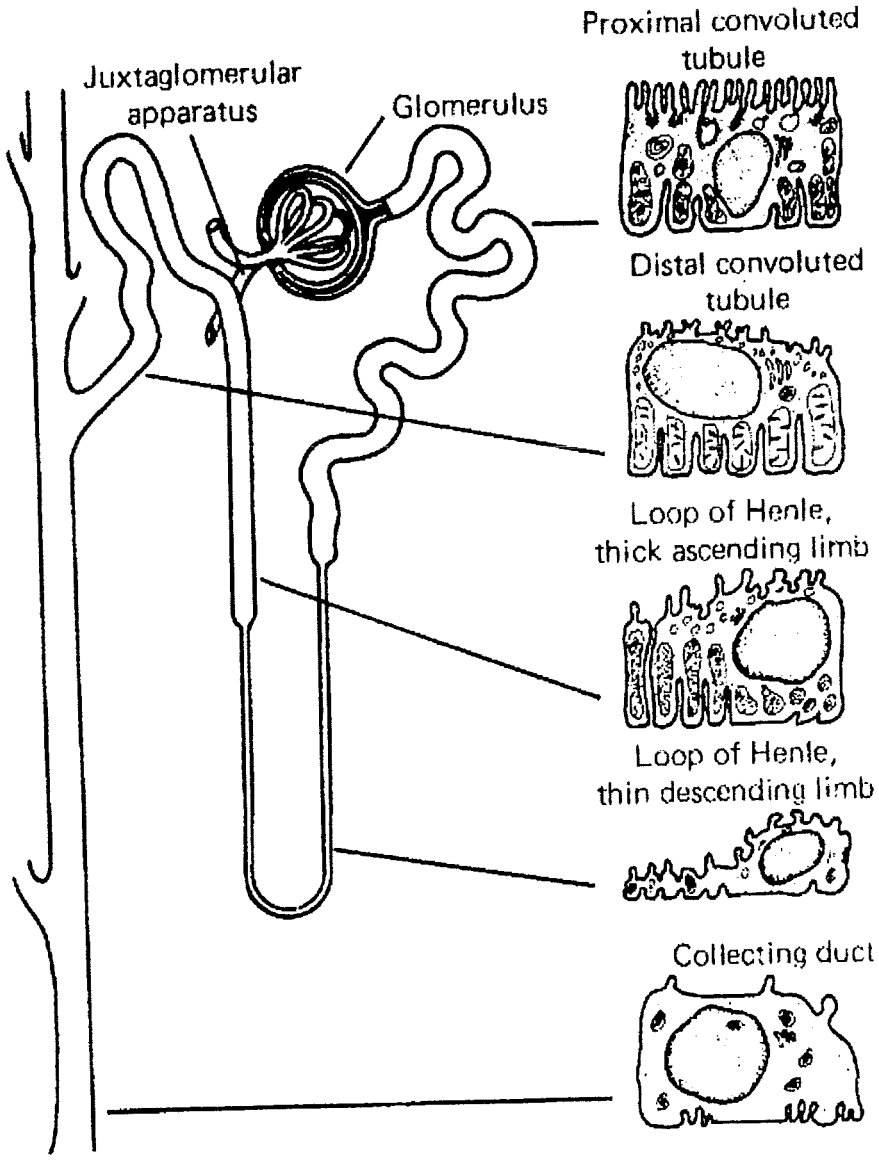


Figure 1.3 The Nephron. The type of cells present in each part of the nephron are shown on the right.[5]

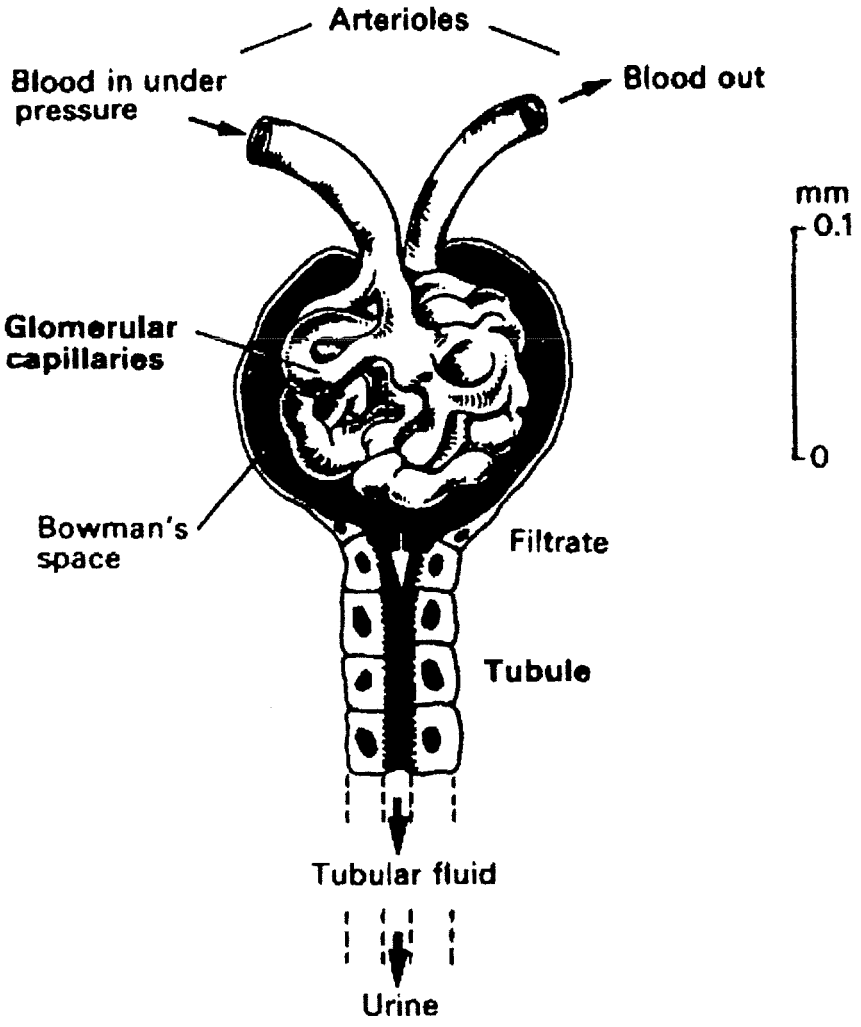


Figure 1.4 The Glomerulus [19]

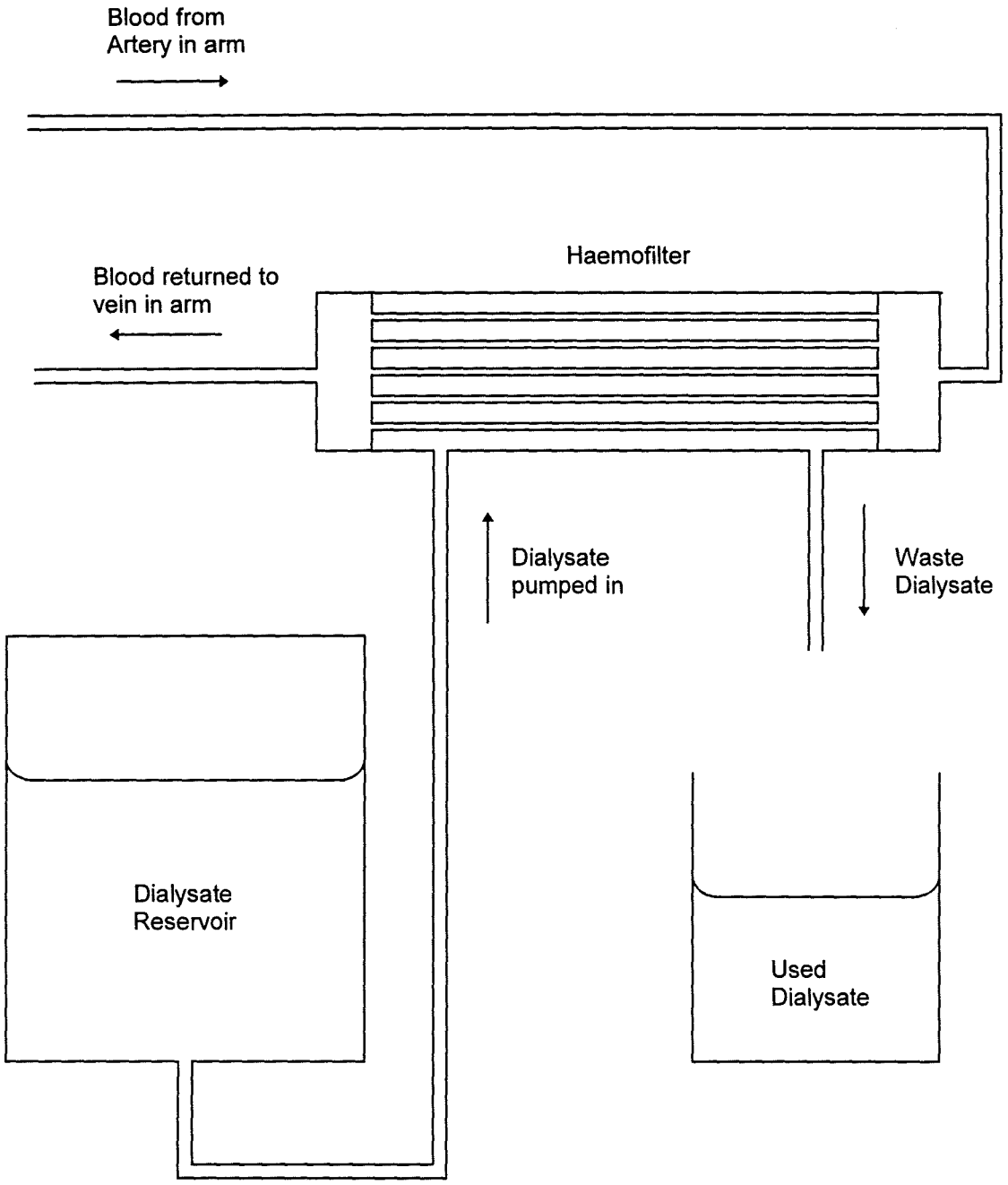


Figure 1.5 The Principle of Haemodialysis

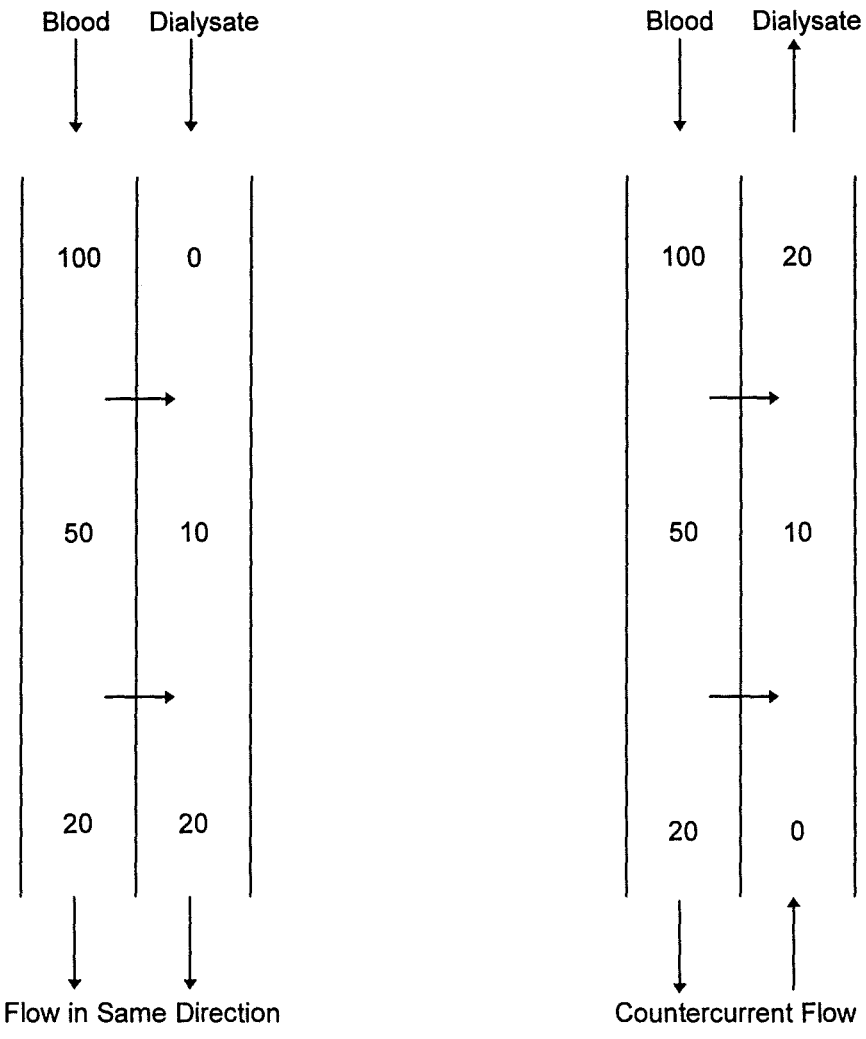


Figure 1.6 The Countercurrent Mechanism

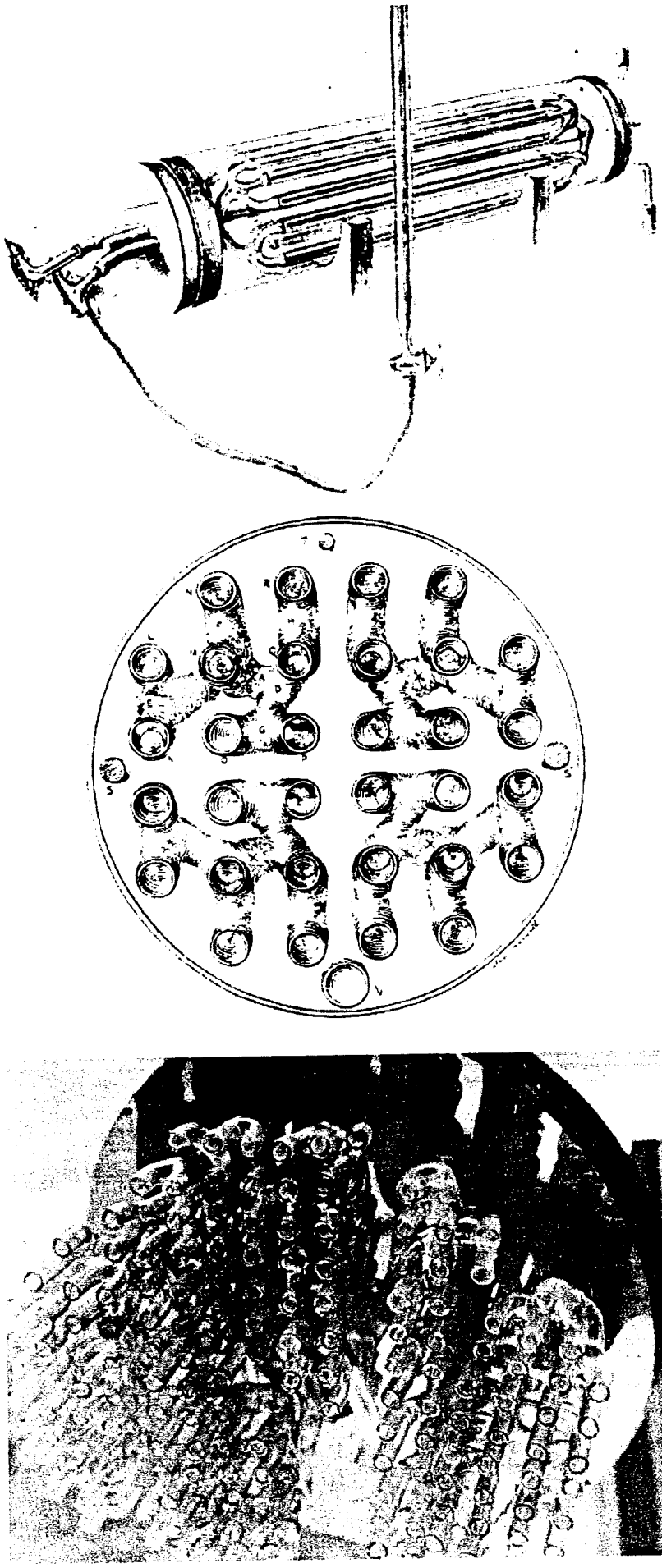


Figure 1.7 Abel's Apparatus [8, 15]

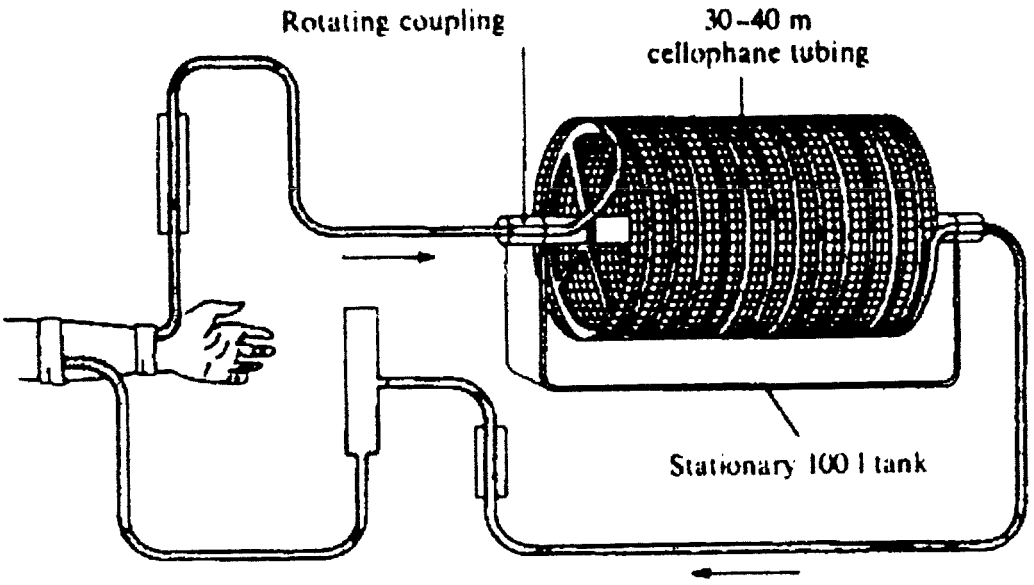


Figure 1.8 Kolff's Rotating Drum Dialyser [8]

CHAPTER 2. LITERATURE REVIEW

Introduction

A wide ranging literature search was undertaken to obtain references relevant to the work being done. This covered both engineering and medical literature. The results of this search are presented here.

2.1 Preterm Birth and Paediatric Renal Failure

There are many medical problems associated with preterm birth [20]. As well as those mentioned earlier, the following conditions are also important :

Patent Ductus Arteriosus. In the womb there is a naturally occurring duct between the right and left sides of the heart, which is part of the foetal circulation. At birth this duct usually closes to establish the new pattern of circulation with the baby breathing through its lungs. If the duct fails to close severe circulatory problems result.

Apnea. This is defined as cessation of breathing for more than 20 seconds. It can result in hypoxia and damage to the central nervous system.

Retinopathy. Damage to the retina can occur due to overexposure to oxygen - this can be a side effect of assisted ventilation.

Hyperbilirubinemia. This condition is defined as higher than normal levels of bilirubin in the blood. It is caused by immaturity of liver function.

Renal function in premature babies is very different from that in adults, and even full term babies [21] . Much of the data used to treat babies under 1 kg is extrapolated from much heavier subjects, and this extrapolation is not necessarily valid.

A few general papers on acute renal failure in neonates have been published [1, 2, 22]. Here the reported incidence in intensive care units ranges between 1 and 8 %. Conservative treatment methods are discussed. These are usually tried first before more invasive methods of treatment such as dialysis are attempted. One conservative measure is known as a fluid challenge. This consists of an infusion of normal saline, given intravenously. The resulting increase in the volume of blood circulating around the body can increase the blood flow through the kidneys, thereby increasing the production of urine. Diuretic drugs such as

frusemide are often given. Diuretics are a class of drug that act on the kidney to increase the output of urine. These and other methods are usually applied before dialysis or haemofiltration are resorted to.

2.2 The Treatment of Renal Failure

There are a variety of extracorporeal treatment options for renal failure once conservative methods have failed [3]. The two main categories are haemodialysis and haemofiltration. Haemodialysis removes substances from the blood by diffusion across a semipermeable membrane down a concentration gradient. This concentration gradient is maintained by the continuous circulation of a specially prepared dialysis fluid on one side of the membrane. Haemofiltration however, removes substances by convection (or bulk movement) across a semipermeable membrane. A pressure difference across the membrane drives the movement - this is called the transmembrane pressure (TMP). There is no dialysis fluid circulating in the haemofilter - the fluid on the other side of the membrane is the ultrafiltrate derived from the blood.

Another factor in the choice of treatment is the method of access to the circulation. The usual route is to take blood from an artery and return it to a vein. For a variety of reasons this is not always possible, in which case the blood can be withdrawn from a vein and returned to a vein, and a pump assisted circuit can be used to provide sufficient blood flow.

The other main choice is between intermittent and continuous treatment. Intermittent treatment can lead to circulatory instability amongst other problems. This is a particularly important problem in paediatric treatment.

These treatment options can be used in a variety of combinations, leading to a large number of permutations, which will be explained in more detail below.

Haemodialysis (HD)

Haemodialysis is used occasionally to treat small infants [23 - 27], but it is not usually the preferred method of treatment, due to its technical complexity. Continuous Peritoneal Dialysis (CPD) is more commonly used. The smallest patient treated with haemodialysis that is mentioned in the literature weighed 2 kg. Both continuous and intermittent treatment have been successful. It is generally agreed that control of fluid removal is the biggest problem with haemodialysis in

such small patients. Babies are much less tolerant than older patients of errors in fluid balance. Bradbury [23] reports the use of linked dialysate pumps to control fluid removal. Pumps are attached to the inflow and outflow of the dialysis compartment of the filter. The pumps are mechanically linked so that they run at identical rates. A separate pump draws off ultrafiltrate at the desired rate. In this way fluid removal can be much more precisely controlled.

It is worth mentioning that haemodialysis is in general more effective than haemofiltration at removing small solutes such as urea. Section 1.4 contains an explanation of the principles of haemodialysis.

Haemofiltration

The principles of haemofiltration have been explained by several authors [28-30]. As in haemodialysis, the patient's blood is passed through a haemofilter. Unlike haemodialysis however, no dialysis fluid is circulated through the outer compartment of the haemofilter. Solute removal is achieved by simple convection across the semipermeable membrane, as opposed to the diffusive process that occurs in dialysis. Convection across the membrane is driven by the pressure differential between the inside and the outside of the semipermeable tubing.

It is usually applied as a continuous therapy for acute renal failure as it is less effective at clearing small molecules (e.g. urea) than haemodialysis. To achieve sufficient rates of ultrafiltration, filters with a high permeability are often used. This technique can be employed either as continuous arteriovenous haemofiltration (CAVH) or continuous venovenous haemofiltration (CVVH).

CAVH

The main advantage of CAVH is its simplicity. No pumps or electronic equipment are needed. Just a haemofilter connected to an arterial and a venous line, and another line to collect the ultrafiltrate. Ronco [31] describes the use of the Amicon minifilter, an example of the type of filter used in this method of treatment. It has a smaller number of hollow fibres, but each fibre has a large internal diameter. The resistance to blood flow is therefore less. This is important as it allows a high blood flow rate through the filter without the need for a blood pump. He used a long ultrafiltrate line arranged to produce a siphoning action on the filter. This creates a negative pressure in the ultrafiltrate compartment, increasing the

ultrafiltration rate that can be achieved. The umbilical vessels provided the access in some of his patients. This is only possible in new born babies, as after 4 or 5 days these vessels close up and become unusable. Lieberman [32] has described a very similar treatment, although he makes no mention of using the ultrafiltrate line to create a siphoning action.

CVVH

Yorgin [33] describes this technique. It is often used when the patient's blood pressure is too low to drive the blood through a CAVH circuit. Blood is withdrawn from a vein and returned to a vein. A blood pump is used to drive the blood around the circuit. Another advantage of this technique is that it avoids the potential complications associated with the cannulation of a limb artery. These include loss of limb circulation and subsequent impaired limb growth.

Mixed Therapies

Dialysis and filtration can be combined into a single therapy [34 - 36]. The basic dialysis circuit is modified so that there is a net removal of fluid from the patient. This can be achieved by having two pumps on the dialysate lines running at different rates, thus producing a negative pressure in the dialysate compartment of the filter. A replacement fluid can then be administered to the patient via the extracorporeal circuit. This fluid is carefully prepared to provide the right balance of solutes needed to achieve biochemical control of the blood composition. It can be infused either before or after the haemofilter. If before the haemofilter, it effectively dilutes the blood, reducing the haematocrit. This leads to an increase in the ultrafiltration rate (see section 2.9).

Other Therapies

Peritoneal dialysis is also used to treat very low birthweight babies [37]. It has been described in section 1.4. This treatment has the advantage of being technically simpler than extracorporeal methods, but there are several associated complications which can often prevent its successful use.

There are a few reports of dialysis being used in conjunction with an extracorporeal membrane oxygenation system (ECMO). This system is used to

oxygenate the blood outside the body when the patient's own respiratory system is unable to supply sufficient oxygen. Summar [38] describes the use of an ECMO system and a dialyser in series to treat hyperammonemia in new born babies. This is a metabolic disorder which results in elevated levels of ammonia in the blood. This and other metabolic disorders are often treated with dialysis therapy.

2.3 Sensors and Monitoring

A large amount of work has been done on the monitoring of the dialysis process, with a view to improving current treatments as well as designing the next generation of dialysis machinery.

One area of particular interest is that of vascular access monitoring [39 - 41]. Various methods have been used to monitor blood access. These include venous pressure monitoring, and measurements of dialysate conductivity to calculate the blood flow rate into the dialyser. This is relevant to the current project because the system uses pressure measurement to determine the adequacy of blood flow. Adequate access is an important problem in adult dialysis, and the problem is even more acute in paediatric dialysis.

Until very recently dialysis systems did not monitor the patient directly. Clinical monitoring was done using separate equipment and nursing staff. New systems are now starting to appear which integrate both aspects of treatment.

One of the main problems with existing systems is the difficulty of giving the correct 'dose' of dialysis. The amount of dialysis to be given is determined at the start of the treatment session, based on measurements of patient weight and other such simple parameters. Because the progress of the dialysis is not directly monitored, it is difficult to give the optimum amount of treatment. A lot of research is directed towards the monitoring of patient urea concentrations during the dialysis session, with the aim of producing a closed loop system that can optimise the amount of dialysis given. Kupcinskas [42] has suggested an optical method that measures the urea concentration in the used dialysate, so that the urea concentration of the blood can be inferred. The system is based on the analysis of infra-red absorption spectra obtained when infra-red light is shone through a transparent tube containing the spent dialysate. Different organic molecules have characteristic absorption patterns, and this allows the concentration of a molecule such as urea to be determined. Canaud [43] has proposed a different method. A quantity of spent dialysate is withdrawn from the dialysis system. The urea it

contains undergoes a chemical reaction which produces ammonium ions in solution (NH_4^+). These ions create a potential difference between two electrodes, which can be measured, thereby determining the urea concentration in the dialysate.

Dialysis can produce a range of unwanted side effects. One of the most serious of these is low blood pressure (hypotension). This is caused by too much fluid being removed by ultrafiltration during the dialysis session. Various systems have been proposed to continuously monitor the amount of fluid being removed. These are based on measurements of haematocrit. This is the percentage by volume of the blood that consists of cells. It is a good indication of the state of hydration of the patient. A high haematocrit implies that the patient is dehydrated (not enough water in the blood), and a low haematocrit means that the patient is over hydrated. A simple method of measuring haematocrit has been described [44]. An infra-red light beam is shone through transparent tubing containing the blood. An infra-red detector measures the amount of light that is transmitted through the blood. As the haematocrit rises, the amount of transmitted light falls. Another method measures the impedance of the blood [45]. The blood is passed through a test cell which is connected into the arterial line of the dialyser. Impedance measurements are taken at a frequency of 5 kHz. The principle of this system is that the blood cells themselves have a greater impedance than the plasma that surrounds them. Therefore, as the haematocrit rises, the impedance also rises.

A study by Jaffrin [46] compares the different methods available for the measurement of haematocrit. It concluded that optical methods are the most accurate and the most convenient to use.

A system designed by Lurzer [47] measures a range of clinical parameters. A PC controlled system can sample blood from the arterial and venous lines, as well as fresh and used dialysate. The sampled blood is fed into a commercial analyser which can measure a range of parameters, such as sodium and potassium concentrations. However, the system is fairly complicated and cumbersome, and it is difficult to see how it could be applied in a clinical system.

2.4 Modelling and Analysis of Renal Replacement Therapy

The current system is designed to be used as a continuous therapy, the treatment continuing for as long as the patient is in acute renal failure. This differs from the treatment given for chronic renal failure - intermittent haemodialysis. This

type of dialysis usually consists of 3 treatment sessions per week, each one lasting several hours.

A fair amount of literature is devoted to the modelling and analysis of the dialysis process. Much of this research is aimed at trying to better understand the dialysis process, with a view to more accurately quantifying the outcome of a given dialysis prescription.

Probably the easiest type of treatment to analyse is continuous arterio-venous haemofiltration (CAVH), as there is no dialysate circuit to consider. As has been mentioned previously, this type of treatment involves the passing of the blood from an artery through a highly permeable haemofilter and back into a vein. The patient's own blood pressure drives the blood through the circuit, and the pressure inside the haemofilter generates an ultrafiltrate, thereby removing waste products from the body. It is important to distinguish between the two different processes that can be used to remove toxic substances from the body. In CAVH, metabolites are removed purely by *convective* movement across the semipermeable membrane. In this process, there is bulk movement of blood plasma through the membrane, driven by a pressure gradient between the inside and the outside of the haemofilter fibre. Dissolved metabolites are therefore removed from the blood stream by convection across the membrane. In haemodialysis however, waste products move across the membrane by *diffusion*. Here, the pressure gradient is irrelevant. It is the concentration gradient between the blood and dialysate that drives the movement of dissolved substances across the membrane. The two processes are often used in combination, as in continuous haemodiafiltration.

Pallone[48] has produced a model of the convective process derived from principles of basic fluid mechanics which gives good agreement with experimental results. This will now be described. The model describes the relationship between the pressures and flows in the components of the extracorporeal circuit using Poiseuille's relationship:

$$\Delta P_i = \frac{-8\, L_i}{\pi\, r_i^4} \mu_b Q_b \qquad (2.1)$$

where ΔP_i is the pressure drop across the ith component, L_i its length, and r_i its internal radius. μ_b is the viscosity of the blood, and Q_b the blood flow rate through the circuit. This form of the relationship is suitable for describing the

tubular components of the circuit, such as the vascular access catheters, and the blood tubing that connects these to the haemofilter. A differential form of the equation is used to describe the pressure variation inside the haemofilter itself:

$$\frac{dP}{dx} = \frac{-8}{N\pi r_f^4}\mu_b Q_b \qquad (2.2)$$

where N is the number of hollow fibres in the haemofilter, and r_f is their internal radius. Using these two equations, the pressures in the extracorporeal circuit can be calculated. In particular, the pressure variation along the length of the haemofilter (which is linear in this model) can be calculated.

The use of the Poiseuille relationship assumes a constant blood viscosity and also that blood is a Newtonian fluid (i.e. that its viscosity does not vary with shear rate). Neither of these assumptions are valid under all conditions. However, Pallone argues that the assumptions are valid under the flow conditions found in CAVH, and other authors of similar studies make the same assumptions. [49, 50]

The modelling of blood viscosity is a more difficult problem, as blood is a complex fluid made up of many different components. Particular factors that affect the viscosity are the haematocrit and the plasma protein concentration. In common with other authors, Pallone uses the empirical relationships of Charm and Kurland [51] to predict viscosity values.

Once the pressure variation inside the haemofilter is known, the volume flux J_v across the membrane can be calculated:

$$J_v = L_p\left\{\left(P - P_f\right) - \Pi_p\right\} \qquad (2.3)$$

L_p is the hydraulic permeability of the membrane. This is an index of its convective properties, and is a function of the membrane material as well as its thickness. P is the local pressure inside the fibre, and P_f is the local pressure outside the fibre (the pressure applied to the ultrafiltrate port of the haemofilter). Π_p is the oncotic pressure. This is an osmotic pressure generated by the protein content of the blood. It tends to draw ultrafiltrate back into the blood, hence the negative sign in equation 2.3. It is a function of the protein concentration in the blood plasma, and can be described empirically by the Landis-Pappenheimer [52] equation:

$$\Pi_p = 2.1\ C_p + 0.16\ C_p^2 + 0.009\ C_p^3 \qquad (2.4)$$

where C_p is the concentration of protein in the plasma.
Volume conservation for blood plasma flow gives the following equation:

$$\frac{dQ_p}{dx} = -J_v \frac{S}{L_f} \qquad (2.5)$$

where Q_p is the local plasma flow rate, S is the total membrane surface area and L_f is the length of the haemofilter fibres. This expression can be integrated along the length of the haemofilter to calculate the total ultrafiltration rate. The relevance of models such as this to the current project will be discussed in chapter 10.

A model of diffusive transport is given by Sargent and Gotch [53]. Diffusion is governed by Fick's law:

$$J = -DA \frac{\Delta c}{\Delta x} \qquad (2.6)$$

J is the flux (mass flow per unit time) of a given solute across a diffusion front of area A. Δc is the incremental change in the concentration of the solute over the incremental distance Δx, perpendicular to the diffusion front. D is the constant of proportionality. In the case of a dialyser membrane, Δx is the membrane thickness, and can be regarded as constant. Therefore, equation 2.6 can be rewritten as:

$$J = -K_0 A \Delta C \qquad (2.7)$$

where K_0 is known as the mass transfer coefficient, a property of the dialyser itself. In this equation, ΔC represents the mean concentration difference across the dialyser membrane. Figure 2.1 shows the solute concentrations in both blood and dialysate as a function of distance along the haemofilter in countercurrent flow. In this model the concentration changes along the haemofilter length are assumed to be linear. The blood enters the filter on the right and its solute concentration falls during passage through the filter. The dialysate enters on the left and the concentration rises from left to right. The centre line represents the difference in

concentration between the two fluids. This model uses a logarithmic mean of this difference.

Hence, from equation 2.7,

$$J = K_0 A\left[\frac{\Delta C_i - \Delta C_0}{\ln(\Delta C_i / \Delta C_0)}\right] \qquad (2.8)$$

In terms of the inlet and outlet blood and dialysate concentrations, equation 2.8 becomes:

$$J = K_0 A\left[\frac{(C_{Bi} - C_{Do}) - (C_{Bo} - C_{Di})}{\ln\left[(C_{Bi} - C_{Do}) / (C_{Bo} - C_{Di})\right]}\right] \qquad (2.9)$$

Dialysance is a parameter often used to quantify dialysis. It is similar to clearance, but takes account of the fact that the solute can appear in the dialysate as well as in the blood. So whereas clearance is defined as

$$K = \frac{\text{solute mass transfer rate}}{\text{concentration of solute in plasma}} \qquad (2.10)$$

dialysance is defined as

$$D = \frac{\text{solute mass transfer rate}}{\text{concentration driving force}} \qquad (2.11)$$

where the concentration driving force is the *difference* in concentration of the solute between the blood and the dialysate. The mass transfer rate is the difference between blood solute concentrations at the inlet and outlet of the dialyser multiplied by the blood flow rate into the dialyser, so equation 2.11 can be written as:

$$D = \frac{Q_{Bi}(C_{Bi} - C_{Bo})}{C_{Bi} - C_{Di}} = \frac{Q_{Di}(C_{Do} - C_{Di})}{C_{Bi} - C_{Di}} \qquad (2.12)$$

where the mass transfer rate is firstly expressed in terms of flows and concentrations on the blood side of the haemofilter, and secondly in terms of parameters on the dialysate side.

Since the mass transfer rate is the same as the solute flux, J, across the dialyser, equation 2.11 can be rearranged as:

$$J = D(C_{Bi} - C_{Di}) \qquad (2.13)$$

Combining 2.13 and 2.9 gives:

$$D(C_{Bi} - C_{Di}) = K_0A\left[\frac{(C_{Bi} - C_{Do}) - (C_{Bo} - C_{Di})}{\ln[(C_{Bi} - C_{Do}) / (C_{Bo} - C_{Di})]}\right] \qquad (2.14)$$

From equation 2.12, expressions for the inlet and outlet concentration differences can be obtained in terms of dialysance, concentration driving force and blood and dialysate flow rates:

$$C_{Bi} - C_{Do} = (C_{Bi} - C_{Di})\left(1 - \frac{D}{Q_D}\right) \qquad (2.15)$$

$$C_{Bo} - C_{Di} = (C_{Bi} - C_{Di})\left(1 - \frac{D}{Q_B}\right) \qquad (2.16)$$

Substitution of these concentration differences into equation 2.14 and rearrangement gives:

$$K_0A = \frac{Q_B}{1 - Q_B / Q_D}\ln\left(\frac{1 - D / Q_D}{1 - D / Q_B}\right) \qquad (2.17)$$

This is an expression for the mass transfer coefficient - membrane area product in terms of blood and dialysate flows and dialysance. The product K_oA is a constant for a given dialyser. Once this constant has been determined, equation 2.17 can be rearranged to predict dialysance in terms of blood and dialysate flows:

$$D = \frac{\exp\left(K_o A\left(1 - Q_B / Q_D\right) / Q_B\right) - 1}{\frac{\exp\left(K_o A\left(1 - Q_B / Q_D\right) / Q_B\right)}{Q_B} - \frac{1}{Q_D}} \qquad (2.18)$$

This is a useful way of predicting dialysance (or clearance) from straightforward measurements of flow rates of blood and dialysate. The relevance of this model to the current project is discussed in chapter 11.

Diffusive and convective transport are often combined in the same treatment. This method of therapy is called continuous arteriovenous haemodialysis by some authors [54] , and continuous arterio-venous haemodiafiltration by others [55]. It is a more complex process due to the interaction of the two different transport processes and therefore it is more difficult to model. It involves the addition of a low flow rate dialysate circuit to a CAVH system. This results in a marked increase in the removal rate (clearance) of middle weight molecules such as urea. Pallone [56] has conducted in vitro experiments and produced a simple model, in an attempt to quantify the clearances achieved in CAVHD. It was shown that at low dialysate flow rates, there is almost complete equilibration of urea between the blood and dialysate, so that the dialysate leaves the filter with almost the same concentration as the incoming blood. Akcahuseyin and Vincent [55, 57 - 59] have developed more sophisticated models, which again compare well with experimental data. This work shows that there is a significant fall off in filter performance after the circuit has been running for several days. This is due to the build up of a protein layer on the semipermeable membrane. Brunet [60] has conducted clinical experiments to quantify the clearances obtained under a range of conditions.

Olbricht [50] has studied the effects of different types of vascular access and filter design on CAVH. He concludes that polyamide filters are more suitable than those made from polysulphone, as they produce a higher filtration flux.

Davenport [61] has investigated the effect of the direction of dialysate flow in CAVHD. It is generally accepted that a countercurrent flow of dialysate is more efficient than a concurrent one, as it leads to a greater mean concentration gradient across the semipermeable membrane. He showed that countercurrent flow produces roughly a 28 % increase in clearance over concurrent flow, all other parameters being equal. The current system alternates between concurrent and

countercurrent flow as the blood is moved back and forth between the two syringes, since the dialysate flow through the filter is always in the same direction. While this is not ideal, the loss of efficiency can be overcome by using dialysate flow rates that are significantly higher than the blood flow rate through the filter.

Research in intermittent haemodialysis aims to increase the efficiency of the treatment process and to optimise the available resources. Since such a large number of people are on maintenance haemodialysis for many years at a time, economic factors are more important than in the care of acute renal failure. Dialysis prescription is not yet a precise science - a lot of research is aimed at producing better models of how the body reacts to dialysis treatment.

One of the major problems is that of fluid removal. Since the patient is unable to regulate body water themselves, dialysis must do this for them, as well as removing waste products. It is difficult to determine the ideal amount of fluid to remove during a dialysis session. If too little is removed, the patient will remain fluid overloaded. If too much is removed, there is a danger of the blood pressure dropping dangerously low. Winnett [62] and Chamney [63] have both worked in this area. Chamney describes the use of blood volume monitoring sensors to produce a closed loop control system. This is much safer than simply prescribing an amount of fluid to be removed at the start of the session. If the patient's total blood volume can be continuously monitored during the dialysis session, then any adverse reaction to the treatment can be detected much earlier.

Similar problems exist with regard to urea removal during dialysis [64, 65]. Sternby describes the single pool model for urea transport. This regards the body as a single volume containing a uniform concentration of urea. This is the model often used for dialysis prescription. The true physiology is more complex than this. Urea exists at several different concentrations in different compartments in the body. These include the blood, the intracellular space and the extracellular space. Urea concentrations in the blood therefore do not fall off in a simple exponential way during dialysis. Better models have been developed to more accurately describe the movement of urea during dialysis.

Vaussenat [66] has investigated the changes that occur in membrane permeability during the dialysis session, using a data acquisition system attached to the dialysis machine. He showed that higher ultrafiltration rates lead to a bigger build up of the protein coat, leading to a loss of membrane permeability.

2.5 Control Systems

The first dialysis machines had very simple control systems. The system of machine and patient was an open loop - there was no feedback of data from the patient to the machine, so the machine could not respond directly to any changes in patient variables. One of the major early problems was control of ultrafiltration. The amount of fluid that would be removed during a dialysis session could not be accurately predicted. This can lead to major clinical problems. If too much fluid is removed the patient's blood pressure can drop dangerously (hypotension). If too little is removed the patient will be fluid overloaded (oedema) leading to high blood pressure (hypertension) and this also can be very dangerous. For an otherwise healthy adult, a normal blood pressure would be in the region of 140/90 mm Hg. A reading of 90/60 mm Hg or below would be considered dangerously low, and 160/95 mm Hg or above would be too high. [67] The first figure represents the systolic pressure, (the arterial pressure when the heart is contracting) and the second figure the diastolic pressure (when the heart is between contractions).

In the 1980s machines that could accurately control ultrafiltration were introduced, and the problems of fluid removal were reduced. The amount of fluid to be removed could be entered into the control system at the start of the dialysis session.

However, such systems still employ open loop control, and the machine cannot respond directly to physiological changes in the patient. The last decade has seen research into much more sophisticated control systems which aim to directly monitor and control a range of patient parameters. The advances in control systems have gone hand in hand with the rapid increase in the availability of cheap and powerful computer technology in the last few years. The work by Bengtsson [68] demonstrates the rapid increase in sophistication of the software used in haemodialysis machines. An example of a commercial controller is given by Sternby [69]. He has successfully developed an adaptive control algorithm. This controller allows the machine to use a wide range of different dialysers and still provide accurate control of ultrafiltration. Both flat plate and hollow fibre filters can be used on the same machine.

Work towards a closed loop system has been done by Giove [70, 71]. He suggests that the clinical knowledge of the doctor or nurse can be applied to the control system by the use of fuzzy logic, turning qualitative knowledge into sets of rules that can be used to control patient parameters. A major problem with any

dialysis control system is that each patient responds to dialysis in a different way, the so-called 'subjective biological response'. The application of fuzzy logic to parameters such as blood pressure and fluid status can overcome this problem.

The work by Giove is directed mainly towards control of ultrafiltration and the fluid status of the patient. Other work [72, 73] aims to control a much wider set of patient parameters, including acid-base balance and sodium concentration. Such a system would dramatically improve patient tolerance to dialysis, minimising undesirable side effects.

2.6 Future Developments

As well as the progress in the area of control systems described earlier, work is being done in a number of other areas, to improve haemodialysis systems. One of these has the aim of reducing the size of extracorporeal systems. A degree of size reduction can result in a portable system, which makes dialysis in the patient's own home more feasible. A further reduction could lead to a wearable system that would operate continuously. The ultimate aim of this research would be to produce a system small enough to be implanted in the body.

The kidney is a very complex structure, and any attempt to produce an artificial device that replaces its function while being of a similar size has to overcome this problem. Each kidney contains approximately 55 km of tubing. Models of renal vasculature have been produced, in an attempt to better understand the flow of fluids through the kidney. One such model [74] is based on the concept that the renal vessels go through six stages of branching and size reduction from the level of the renal artery through to the nephrons themselves. The arterial radius at each stage is described by:

$$r_N = r_0 e^{-0.9N} \qquad (2.19)$$

where r_0 is the radius of the aorta, N is the number of the branching stage, and r_N the radius at that branching stage. The blood flow rate, Q_N, through a vessel is given by:

$$Q_N = 2.716\, e^{0.244(N-1)}\, r_N^3 \qquad (2.20)$$

The authors do not say how either of these equations were arrived at. They allow the flow rates in the various vessels of the kidney to be estimated. However, in the absence of any experimental data to compare the results with, it is not possible to assess the validity of this model.

One feasibility study [75] has concluded that an implantable device is technically possible. The authors do not explicitly state how they reached this conclusion. However, they do concede that many very difficult technical and medical problems would need to be overcome.

On a more practical level, a lot of work has been aimed at reducing the size of dialysis equipment, to make the systems more portable. The major obstacle to portability is the need for large quantities of treated water (around 600 litres per week) for the preparation of dialysis fluid. If the dialysis fluid can be recycled then the system can be made portable. Mourad [76] describes a system that does this (see figure 2.2). Only 10 litres of dialysate is needed. The fluid is recirculated through a cartridge that contains urease and activated carbon amongst other things. Urease is an enzyme that catalyses the conversion of urea to ammonia. The cartridge reduces the concentration of waste substances in the dialysate so it can be recirculated through the dialyser. A similar system has been described by Bigsby [77]. Here just activated carbon is used to absorb the waste substances. Both systems were found to give sufficient clearance to be of practical use. Portable systems such as these are already commercially available. However a 'wearable' system which allows the patient to remain mobile still seems a long way off.

2.7 Assessment of Manual System

The manual system on which this project is based is described by Coulthard [6]. The configuration of the system is illustrated in figure 1.1 (see chapter 1). It is constructed from standard luer locking clinical equipment. The syringes used are of a standard plastic disposable type, either 5 ml or 10 ml in size. Two types of polysulphone hollow fibre haemofilter have been used. The first was an Amicon Minifilter, with a blood priming volume of 6 ml and a membrane surface area of 0.015 m^2. The second was an Amicon non-clinical product with the same priming volume but finer capillary tubing, given a higher membrane surface area of 0.04 m^2.

The apparatus operates as follows. Blood is withdrawn from the patient using syringe 1 through taps B and A. The amount withdrawn depends on the size

of the patient. For the smallest babies this is as little as 3 ml. For the larger babies up to 10 ml may be withdrawn. The rate of blood withdrawal also varies depending on how good the vascular access is. With good access, the rate of withdrawal would be approximately 10 ml/min. The right hand end of the haemofilter is closed off during this operation, as is the heparin infusion line. When the required amount of blood has been withdrawn, tap A is turned so that the heparin infusion can continue into the venous line, and tap B is turned so that the venous line is cut off from the rest of the circuit and the right hand end of the haemofilter is now open. Blood filtration can now begin. The plunger of syringe 1 is depressed to push the blood through the haemofilter into syringe 2. This is done fairly slowly, aiming for a blood flow rate of approximately 10 ml/min. The elastic band attached to syringe 2 provides the necessary transmembrane pressure to allow ultrafiltration to take place. The mean pressure generated by the elastic band is approximately 300 mm Hg. When all the blood is in syringe 2, the plunger of syringe 1 is released, allowing the blood to flow back into syringe 1. The cycle is repeated until the required reduction in blood volume has been achieved. With all the blood in syringe1, tap B is turned again to close off the haemofilter and open access to the venous line. Tap A is turned to close off the heparin line and open the venous line. The filtered blood is then returned to the patient, again aiming for a 10 ml/min flow rate. The whole process can then begin again.

Elastic bands of different strengths can be used to give different transmembrane pressures and therefore different filtration rates. The disadvantage of this technique is that the pressure generated depends on the extension of the elastic band and therefore on the position of the plunger of syringe 2. However this is not a serious disadvantage in a system such as this where the amount of ultrafiltrate obtained in each operating cycle is being measured manually anyway. The possibility of a breakage is also not a serious problem. It would not result in any serious harm to the apparatus or to the patient. The elastic band would simply have to be replaced.

Heparin is infused into the apparatus via tap C, using a standard infusion pump. Other intravenous infusions can also be given via tap C. At the end of the treatment session, the blood that remains in the haemofilter can be flushed back into the patient by replacing it with saline infused via tap E.

Three case reports of the system's use are given. The weights of the babies treated were 808 g, 630 g and 1140 g. In each case, the system achieved sufficient ultrafiltration and biochemical control of the blood plasma. Although each

of the babies died, this was not due to the failure of the system. Their renal failure was secondary to other more serious conditions. The system worked well in these cases.

In developing the automated system, several improvements to the original manual apparatus were made. These are dealt with in chapter 4.

2.8 Work of a Similar Nature

The current project represents a totally new approach to the treatment of paediatric renal failure. An extensive literature search has failed to find any work based on the same principles. However, two systems were found that bear some superficial resemblance to the current work.

In 1985, De Virgiliis [78] designed a dialysis system that used a 30 ml syringe as the blood pump (see figure 2.3). It was an otherwise conventional blood circuit that replaced the normal peristaltic pump with a syringe pump. The syringe was driven by a crank and rod mechanism. The type of motor used is not specified - but its speed could be controlled to give different blood flow rates. The blood is pumped in one direction around the circuit, with the aid of three tube clamps which open and close in time with the syringe pump. Several advantages are claimed for this system over a conventional design. The syringe pump allows the blood flow rate in the circuit to be more accurately determined than with a peristaltic pump. The author also claims that haemolysis of red blood cells is significantly reduced, although the data he quotes is rather vague.

The Ariadne 1 was a new type of single needle dialysis system developed in Belgium in the 1980s [79] (see figure 2.4). It is similar to the current project in that blood flow in the dialyser is bi-directional. Blood is pumped from the patient through the dialyser into a collecting bag that is attached to a weigh scale. When the weight of blood in the bag reaches a pre-set level the roller pump reverses direction and the blood is pumped back through the dialyser and back into the patient. The machine was intended for home dialysis, designed to be compact and easy to use. Its use for ultrashort daily dialysis is described by Hombrouckx [80]. This treatment regime involves dialysis sessions of only 1.5 hours 6 times weekly, as opposed to a more conventional regime of 4 hours 3 times weekly. The reasoning behind this is explained by the 'double pool' model of urea distribution in the body. Urea can be regarded as being contained in two separate pools, the extracellular and the intracellular. The extracellular pool is mainly the circulation. The

intracellular pool is the urea that is contained inside the cells of the body. During dialysis the urea concentration in the blood drops rapidly during the first hour of treatment. This results in a concentration gradient between the two pools. Urea diffuses from the intracellular space to the extracellular space. However this process is slow, and equilibrium is only achieved after several hours. So in the meantime the urea concentration in the blood is low and dialysis will be less efficient during the latter stages of a 4 hour treatment session. Therefore, better urea clearance will be obtained with more frequent but shorter dialysis sessions.

2.9 Miscellaneous Literature

Continuous haemodiafiltration is one of many different methods of renal replacement therapy in common use. In essence it involves the removal of fluid from the patient by ultrafiltration and the substitution of this fluid by the infusion of a suitable replacement fluid into the dialysis circuit, either before or after the haemofilter. The advantage of this treatment is that much larger clearances can be achieved than with conventional ultrafiltration and/or dialysis techniques. Patient access can be via an artery and a vein (CAVHD) or just a vein (CVVHD). The use of this technique had been suggested in connection with the current project, and the feasibility of this was investigated. There are two major problems. The first is the accuracy needed in the preparation of the substitution fluid. Very small errors in the concentration of the various components of the fluid can lead to very large swings in patient biochemistry, causing the patient to become ill very quickly. The other problem is that of fluid delivery. The fluid has to be delivered at very precise volume flow rates, and the intravenous pumps currently available are not capable of this kind of accuracy [81, 82] . Both piston type and peristaltic pumps are subject to the same problem. The inaccuracies reported are as high as 10 %. This would be completely unacceptable in a preterm baby. Another relevant observation has been made by Kameneva [83]. He reports that dilution of blood with substitution fluid leads to a significant increase in haemolysis. It appears that plasma proteins have a protective effect on red blood cells, and when the protein concentration is reduced the blood cell damage increases noticeably. In view of these problems it was decided not to pursue the idea of haemodiafiltration.

A typical patient may need haemodialysis for up to one week. It is known that the performance of the haemofilter membrane degrades over time. In particular, protein builds up on the membrane surface, leading to a decrease in

permeability. The loss of efficiency has to be balanced against the inconvenience and increased risk of infection associated with a change of haemofilter. Schaeffer [84] investigated this problem. His study showed no significant loss of performance in the first 72 hours of treatment. This would imply that only one change of filter would be necessary for a total treatment time of one week.

Summary

The relevant medical literature was presented in sections 2.1 and 2.2. This concentrated on paediatric renal failure and its treatment. Relevant engineering literature was reviewed in sections 2.3 to 2.6. The manual system that was the basis for the current project was reviewed in section 2.7. The literature search failed to find any evidence of similar systems to the one being developed. Systems bearing a superficial resemblance were described in section 2.8. Finally, section 2.9 reviewed some miscellaneous literature that was considered relevant.

Once the literature review was complete, the project could proceed to the specification stage. This is described in the next chapter.

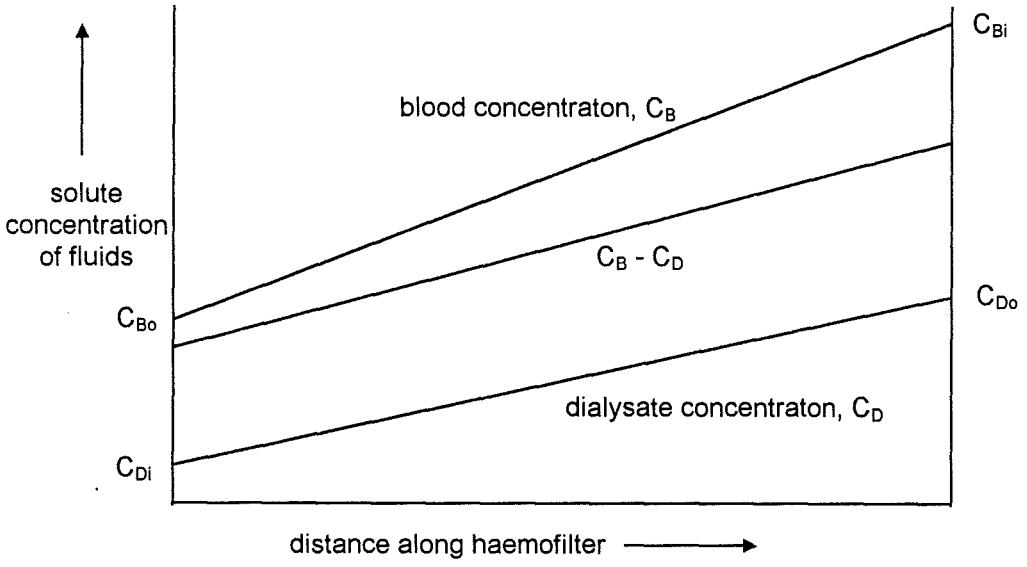


Figure 2.1 Solute Concentration vs. Distance along Haemofilter

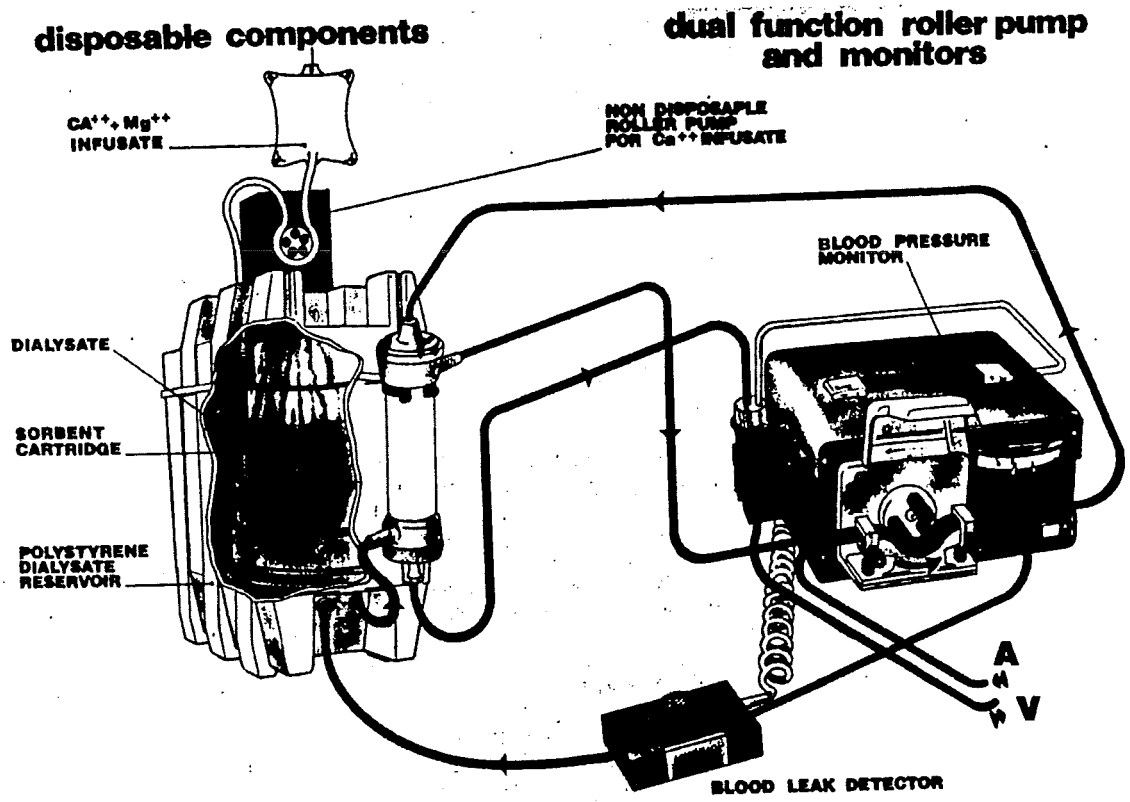


Figure 2.2 A System to Recycle Dialysis Fluid [76]

PAGE

NUMBERING

AS ORIGINAL

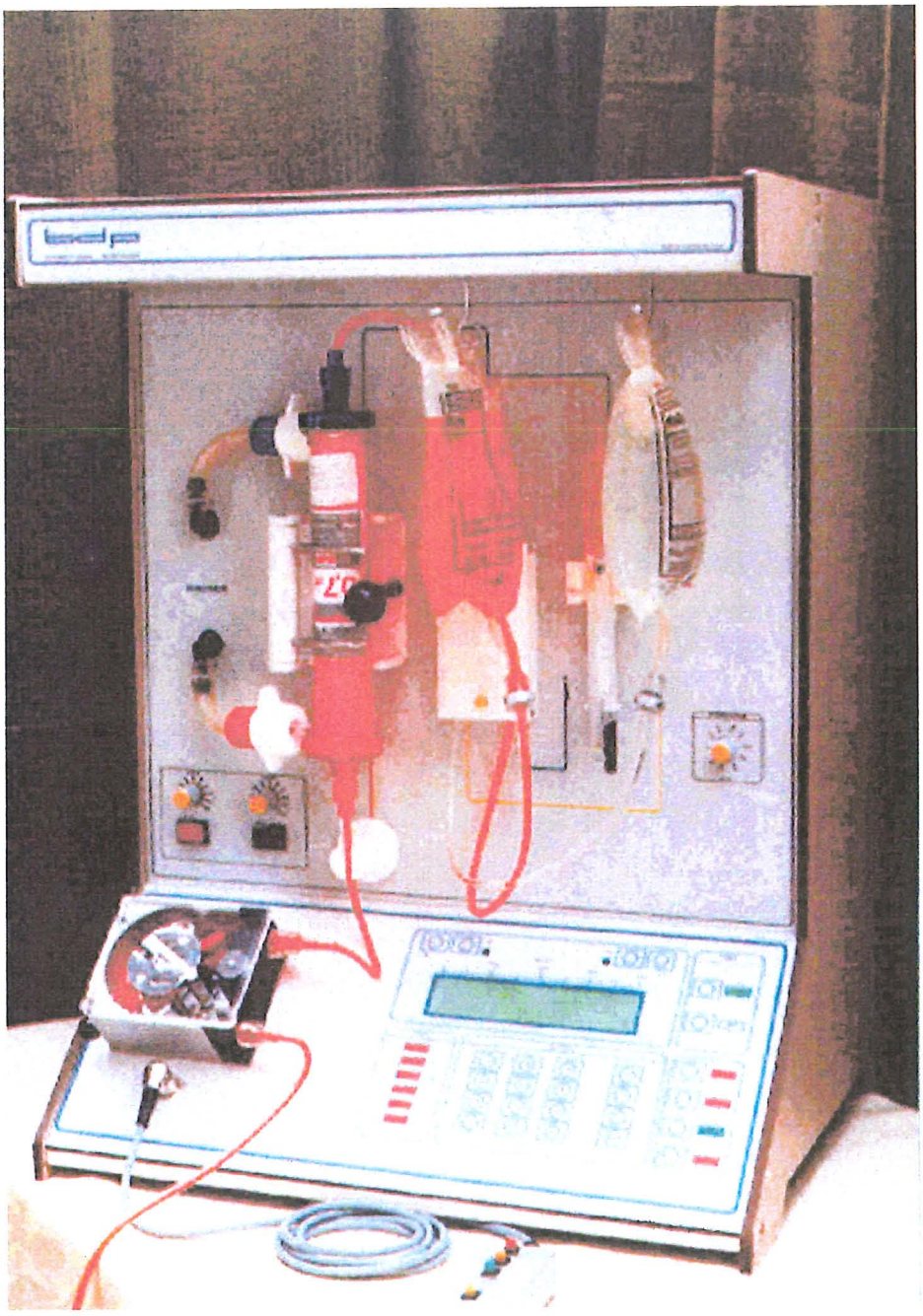


Figure 2.4 The Ariadne 1 Dialysis System [85]

CHAPTER 3. SPECIFICATION OF PROTOTYPE SYSTEM

Introduction

At the start of the project, a specification for the automated system was drawn up in consultation with the clinical staff of the paediatric nephrology department at the Royal Victoria Infirmary in Newcastle. This is presented in detail here, with explanations where necessary.

3.1 Extracorporeal Apparatus

This must be designed so that the total blood containing volume (excluding syringes) is kept to an absolute minimum.

The recirculation fraction of the apparatus must also be kept to a minimum. This is the percentage of the extracorporeal blood volume that passes through the haemofilter more than once. The higher the recirculation fraction is, the lower is the efficiency of the dialysis system.

The apparatus must allow for the controlled delivery of other fluids such as heparin. Access must also be available to allow flushing of the apparatus with saline at the end of each treatment session, so that as much blood as possible is returned to the patient.

3.2 Mechanical Apparatus

It must be straightforward and quick to attach and remove the extracorporeal apparatus to/from the machine.

The apparatus must be able to accommodate different sizes of haemofilter.

The physical layout of the apparatus should be such that the venous line connecting the patient to the haemofiltration system can be as short as possible, to minimise the recirculation fraction and the total volume of the extracorporeal apparatus.

Noise emitted during normal operation should be kept to a minimum.

The apparatus must be easy to clean to minimize the risk of infection to the patient.

3.3 Blood Withdrawal and Return

The system must be able to withdraw and return safely a measured volume of blood from the patient, to an accuracy of 0.1 ml. The minimum volume required is 3 ml, and the maximum is 10 ml.

The blood withdrawal and return system must mimic the procedure used by a clinician to access the patient's blood. The system must be able to sense blockages in the venous catheter and act accordingly. If 0.5 ml of plunger movement results in no blood being withdrawn, then the plunger should stop to allow time for the blockage to clear. If the blockage does not clear, then eventually the blood flow should be reversed, but no more than 0.5 ml of blood should be pushed back down the venous line. The system should be able to respond to blockages by withdrawing blood at a slower rate.

Pressures generated in the venous line must be carefully monitored to prevent any injury to the patient.

The venous line must be monitored to prevent the possibly of an air embolism occurring.

The system should aim to withdraw and return blood at a rate of 10 ml/min. The upper limit for blood return rate is based on the assumption that a 50 % increase in total cardiac blood flow is acceptable. For a 1 kg baby, the total cardiac output is approximately 60 ml/min. So a safe rate of blood return would be 30 ml/min. Therefore a specified rate of 10 ml/min is well below the safety limit.

3.4 Ultrafiltration

The system must be able to supply a range of ultrafiltration rates from 3 ml/h up to 15 ml/h, at an accuracy of ±5 %.

The blood flow rate through the haemofilter should be 10 ml/min during haemofiltration.
Each volume of blood withdrawn from the patient should be haemofiltered for approximately 4 minutes before being returned to the patient.

Back filtration (i.e. the movement of filtrate back into the blood) must be prevented from occurring at all times.

System pressure should be monitored during ultrafiltration so that any blockages in the haemofilter can be detected and dealt with.

3.5 User Interface and Control System

The user interface should be easy to operate.

The main control parameters will be working blood volume and ultrafiltration rate. Working blood volume can be set at the beginning of each treatment session and does not need to be changed during the session. However it should be possible to change the ultrafiltration rate easily during the treatment session.

The user interface should perform a standard calculation of working blood volume based on the weight of the baby entered by the operator. This calculation is based on the assumption that it is safe to withdraw 6 % of the baby's total blood volume. The total blood volume is calculated from the body weight using a figure of 85 ml/kg for the blood volume to body weight ratio.

The user interface should provide information on the current treatment session. This includes the elapsed time since the session began, as well as a running total of the amount of filtrate removed.

It should also provide the same information about previous treatment sessions.

3.6 Safety Requirements

Detection of an air bubble in the venous line must result in the immediate halting of the system and the sounding of an audible alarm.

All system faults must be reported by an audible alarm.

All faults must result in the system stopping in a 'fail safe' condition.

3.7 Other Requirements

The system should be able to operate continuously for periods of up to one week. Continuous haemofiltration is preferable because this minimises variations in the blood biochemistry of the patient.

The system should produce no electromagnetic interference that could affect other equipment being used to treat the patient.

Summary

The clinical requirements of the patient with acute renal failure define much of the specification presented above, in particular the sections dealing with the extracorporeal apparatus, blood withdrawal/return and ultrafiltration. Most of the requirements in the mechanical apparatus section arise from the needs of the clinical staff who would operate the system. This also applies to the user interface and control system section.

Once the specification had been firmly established, initial design work could begin, based on this specification. This is the subject of the next chapter.

CHAPTER 4. INITIAL DESIGN WORK

Introduction

This chapter deals with the major design decisions that were made at the start of the design process. These fell into three main categories, and are detailed below.

4.1 Design of Extracorporeal Apparatus

The design of the clinical apparatus was considered in detail, to see if any fundamental changes could be made to improve it. However, it was decided that the basic arrangement of a haemofilter with a syringe at each end could not be improved upon. Any other arrangement inevitably leads to an increase in the total blood volume of the system, and this has to be avoided at all costs, as required by the specification.

It would be possible to dispense with the 3 way taps that control the flow of blood, and replace these with a solenoid driven tube clamp system. Figure 4.1 illustrates a suitable layout, indicating the positions where the solenoid clamps would need to be placed. One advantage of this system is that the disposable element is simply a piece of tubing, rather than a 3 way tap, thereby reducing the cost of the disposable apparatus. However, a big disadvantage of this arrangement is that the system would be more difficult to operate manually. As detailed in the specification, manual operation is required at the beginning and end of a treatment session. Also, it proved difficult to obtain suitable solenoids commercially. Because of these two problems it was decided to keep the 3 way taps.

The original apparatus functioned well in the clinical tests, but it was decided that several improvements could still be made. The haemofilter used was an Amicon Minifilter with a blood priming volume of 6 ml. A filter with a much smaller priming volume (3.5 ml) became available (Hospal Miniflow 10), so it was decided to use this in preference to the Amicon product. The blood priming volume of the extracorporeal apparatus is probably the single most important parameter of the whole system, so any reduction in this is a significant improvement in the system. The volume of the 3 way taps and lines is 0.34 ml, so the new filter

represents a 40 % reduction in the priming volume (ignoring the volume contained in the syringes).

The two syringes were positioned on the same side of the haemofilter to reduce the overall size of the machine. The original configuration of the three moving taps A,B and C (see figure 1.1) was reduced to two with no loss of functionality, and a reduction in circuit volume. This reduces the mechanical actuation requirements significantly. A pressure transducer was added to allow pressure monitoring within the apparatus. The final configuration shown in figure 4.2 was arrived at.

4.2 Mechanical Actuation of Syringes

Several methods of driving the syringes were considered. There are two main alternatives - a crank and connecting rod or a lead screw system. Either method is feasible but the lead screw is less complex mechanically and is the obvious choice for this application. The lead screw would be driven through a gearbox with an appropriate reduction ratio - the ratio would obviously be chosen in combination with the choice of the pitch of the lead screw.

Figure 4.3 shows one possible design employing a lead screw system. Here the syringes are mounted on opposite sides of the haemofilter in a 'horizontally opposed' configuration. Since the motion required during the filtration phase is reciprocal, this configuration allows both syringes to be driven by a single motor. However, at certain times during the operating cycle the syringes need to be driven independently, so this is not a practical solution.

Figure 4.4 illustrates another possible design. The right hand syringe is driven by a lead screw and motor. The elastic band on the left hand syringe of the original system is replaced by a spring.

The transmembrane pressure (i.e. the pressure difference between the blood and the filtrate in the haemofilter) might be fairly hard to control with this design. This is because the spring will not exert a constant force on the syringe plunger - so as the blood enters the left hand syringe the pressure in that syringe will increase linearly (force proportional to extension in a spring) so the absolute pressure of the blood will not be constant throughout the passage of the stroke volume through the haemofilter. As transmembrane pressure is a critical parameter in the filtering process this is a serious disadvantage.

It might be possible to overcome this problem by using a spring that is a lot longer than the stroke length of the syringe. Then the force exerted by the spring would be approximately constant throughout the travel of the syringe plunger.

Figure 4.5 shows a more advanced design. Here the spring/elastic band idea on the left hand syringe has been dispensed with altogether. Here a lead screw/motor combination drives both syringes.

Replacing the spring driven plunger with a motor driven one allows for greater flexibility in the control of the blood flow and pressure.

The pressure in the blood circuit, and therefore the ultrafiltration rate, could be directly controlled by the two syringe drivers (see Figure 4.6). At the beginning of the filtration phase, syringe A drives down while plunger B remains stationary until the required TMP has been reached. The two plungers then follow each other precisely as the blood is pumped through the haemofilter, thereby maintaining a constant TMP. This pressure would need to be constantly monitored and small adjustments made to the relative motion of the two syringes to maintain a constant pressure. The control algorithm would be fairly straightforward as at the end of each pumping cycle the plungers would swap 'logical places' and plunger A would then follow plunger B.

After much consideration this was the configuration that was settled upon.

4.3 Control System

Various control strategies were considered. The main choice was between PC based control and a more dedicated system, e.g. a programmable logic controller. Eventually PC control using a data acquisition card was chosen. This has two main advantages over a PLC based system.
A PC system is flexible and powerful, as a high level programming language can be used to provide control. Also, a graphical user interface can be used to control the machine, greatly increasing the ease of operation.

4.3.1 *Control Hardware*

Once the PC had been chosen as the platform for the control system, it was necessary to select a suitable data acquisition board to provide the interface between the machine and the computer. Boards are specified mainly in terms of the number of digital and analogue data lines that they provide.

The requirements of the board were as follows:

Digital Output Lines	Number of lines
2 Stepper Motors (each motor has 1 clock and 1 direction line)	4
2 3 Way Tap Drivers (2 lines for each motor - forward and reverse)	4
Total:	8

Digital Input Lines

2 Stepper Motor position microswitches	2
4 photodetectors	4
Total:	6

Analogue Input Lines

Pressure Transducer	1

Analogue Output Lines

None

So, the total requirement was for 8 digital output lines, 6 digital input lines and 1 analogue input line. Boards from Keithley and National Instruments were considered. The National Instruments Lab-PC-1200AI was chosen as the lowest cost board that could fulfil the requirements. This is a relatively slow ISA device. However, it is easily fast enough for this application, as there is no need for high rates of data transfer. It has 24 digital lines that can be configured as either input or output, so there are 10 spare lines. It has 8 analogue input lines. There is no analogue output capability, which makes it less expensive than the standard 1200 card.

When the project began, a Pentium 233 MHz PC with 32Mb RAM was already available. The data acquisition card was installed in this computer. It was used throughout the development of the prototype, and was found to have

sufficient performance for the application. In the final prototype system, a different PC was used, with very similar specification, apart from a slightly slower processor speed of 200 MHz.

4.3.2 *Control Software*

The National Instruments card comes complete with its own driver software, NI-DAQ. This software controls all the low level functions of the card. Another level of software is needed above this, so that a control system and user interface can be written. Two National Instruments products were suitable candidates, LabView and LabWindows/CVI. LabView has a more graphical approach to programming, whereas LabWindows is based on C. Labwindows/CVI was preferred because of the flexibility and degree of control that programming in C gives.

It is a Windows based programming package that contains several function libraries that help to speed up program development. A graphical user interface can be easily constructed using a menu based editor. This interface can then be linked to underlying code using 'callback' functions, which can be attached to each object on the interface. The underlying code (the control software) is programmed in C. The package is primarily designed for the production of virtual instrumentation systems.

Summary

The design alternatives that were considered are summarised in tables 4.1, 4.2 and 4.3. Each table is a decision matrix that attempts to quantify the design options that were considered. The first row contains relevant design criteria, together with a number between 1 and 10 to indicate their relative importance in the decision making process. Each design alternative is then assigned a number (also between 1 and 10) to indicate how well it satisfies a particular criterion. This number is then multiplied by the relative importance of the criterion, and the results are summed across the columns to give the totals in the right hand column. This final figure gives an indication of the relative merits of each design alternative.

Once the major design decisions had been made, the detailed design of the system could begin. This commenced with the mechanical and electromechanical systems, which is the subject of chapter 5.

	Ease of manual operation (8)	Commercial availability (8)	Simplicity of design (5)	Quietness of operation (5)	Size of device (4)	Cost saving for disposable items (2)	TOTAL
Solenoid driven tube clamp	2	1	7	3	7	6	114
Servomotor 3 way tap driver	8	7	3	6	4	4	189

Table 4.1 Blood Flow Control

	Inherent accuracy (8)	Reliability (8)	Simplicity of control algorithm (6)	Cost saving (4)	Development time (4)	Simplicity of design (4)	TOTAL
Lead screw	6	5	6	6	6	8	204
Crank & rod	4	4	4	2	2	4	120

Table 4.2 Syringe Actuation

	Functionality (8)	Accuracy (8)	Control of TMP (8)	Reliability (6)	Simplicity of design (5)	Cost Saving (5)	TOTAL
1 Motor	2	7	2	7	8	8	210
1 Motor + spring	5	4	5	5	6	8	212
2 Motors	7	7	8	4	4	5	245

Table 4.3 Syringe Driver Design

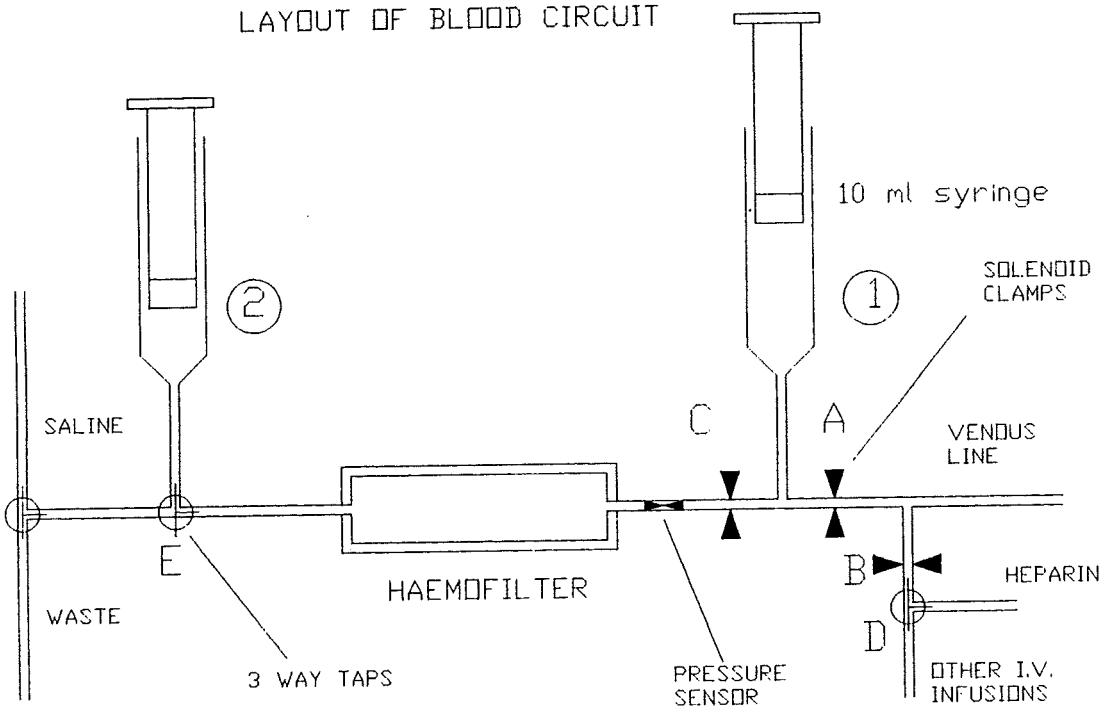


Figure 4.1 Tube Clamp Driven System

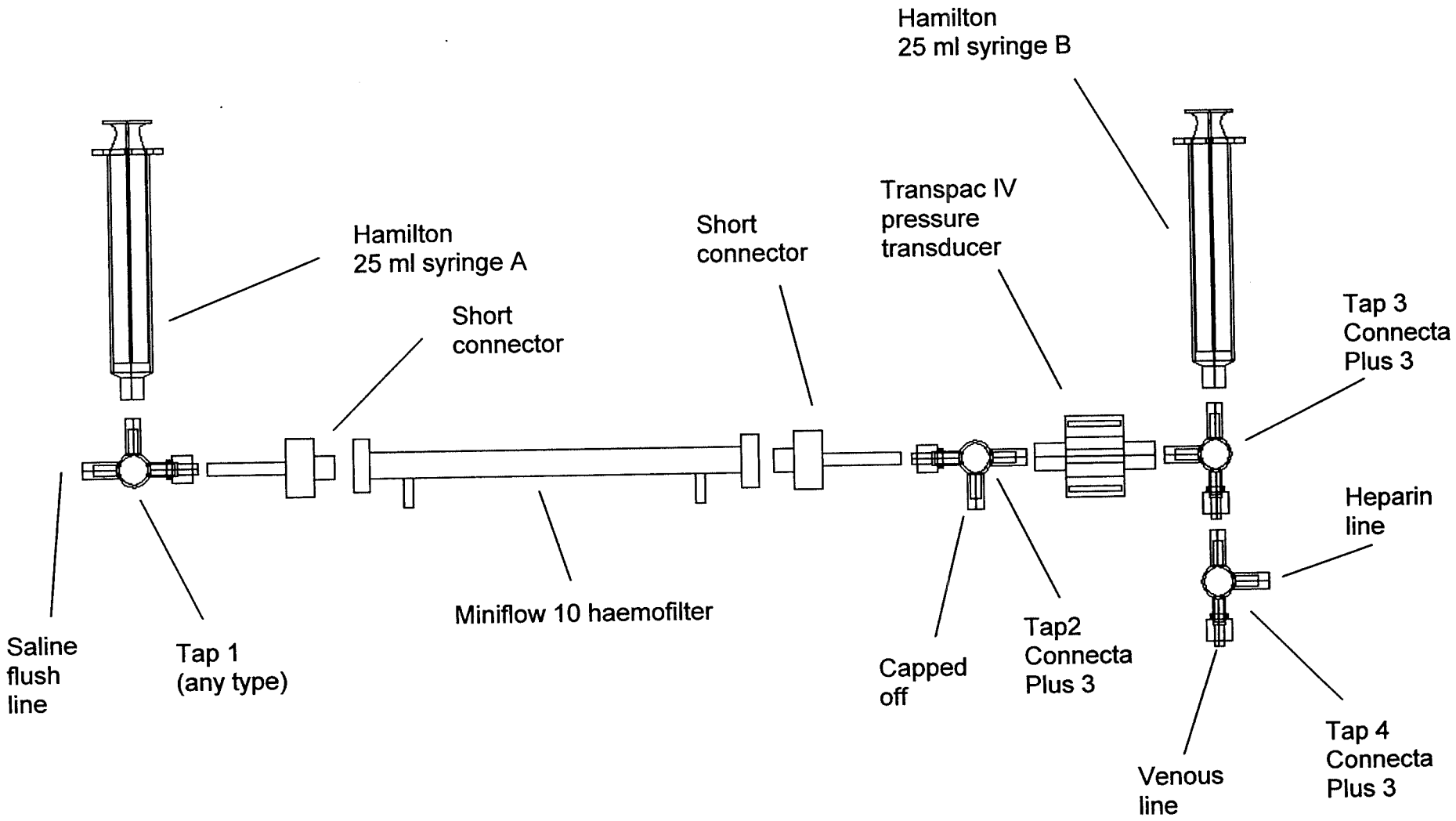


Figure 4.2 Final Configuration of Circuit

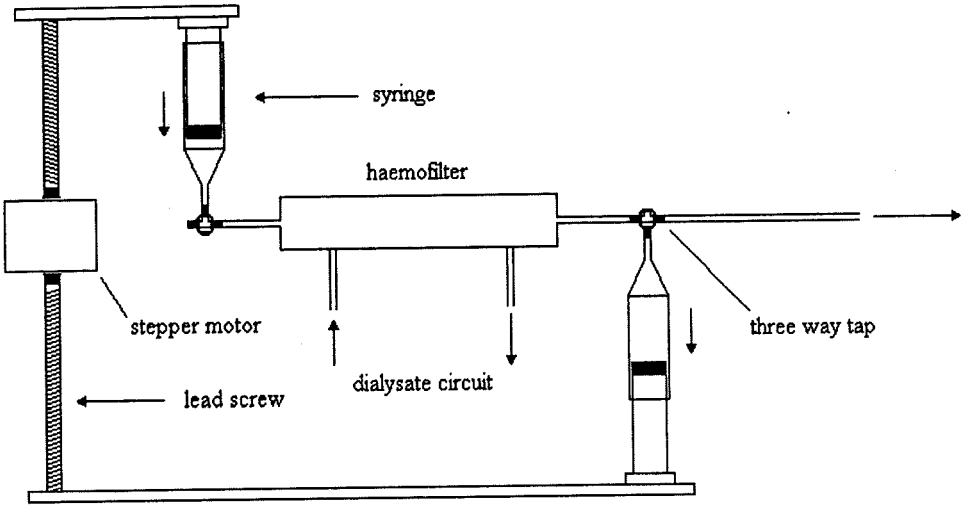


Figure 4.3 Horizontally Opposed Configuration

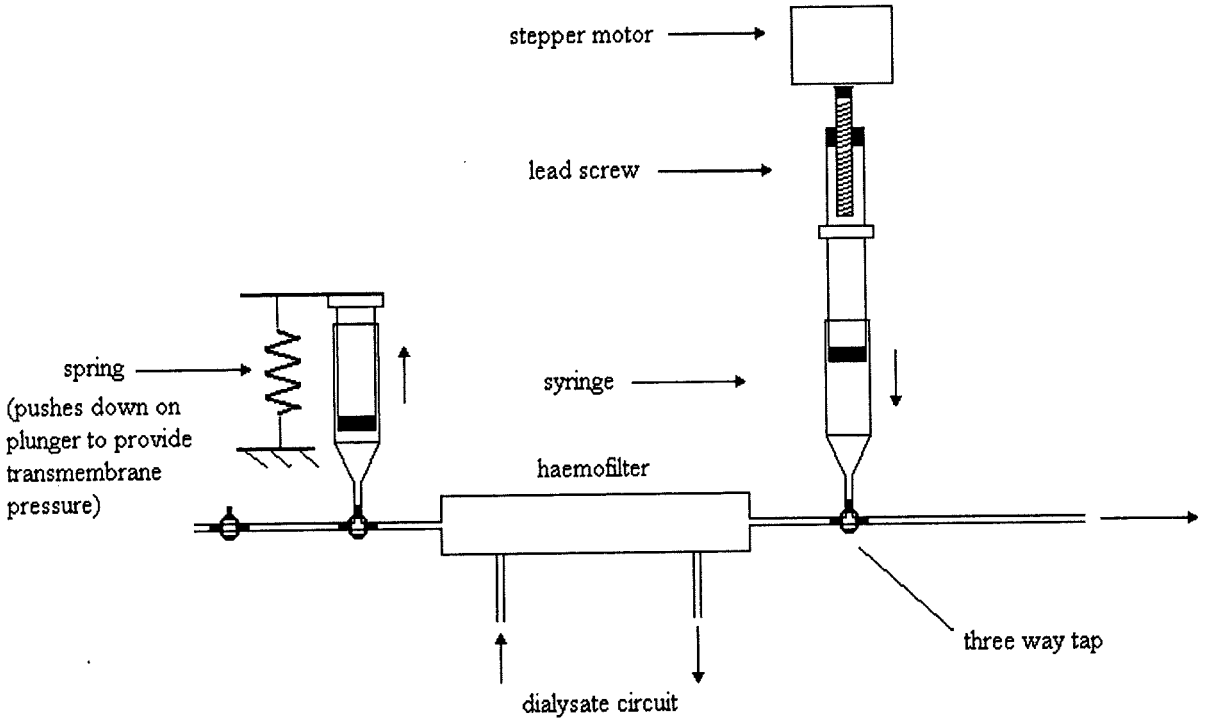


Figure 4.4 Two Way Circuit using One Syringe Pump

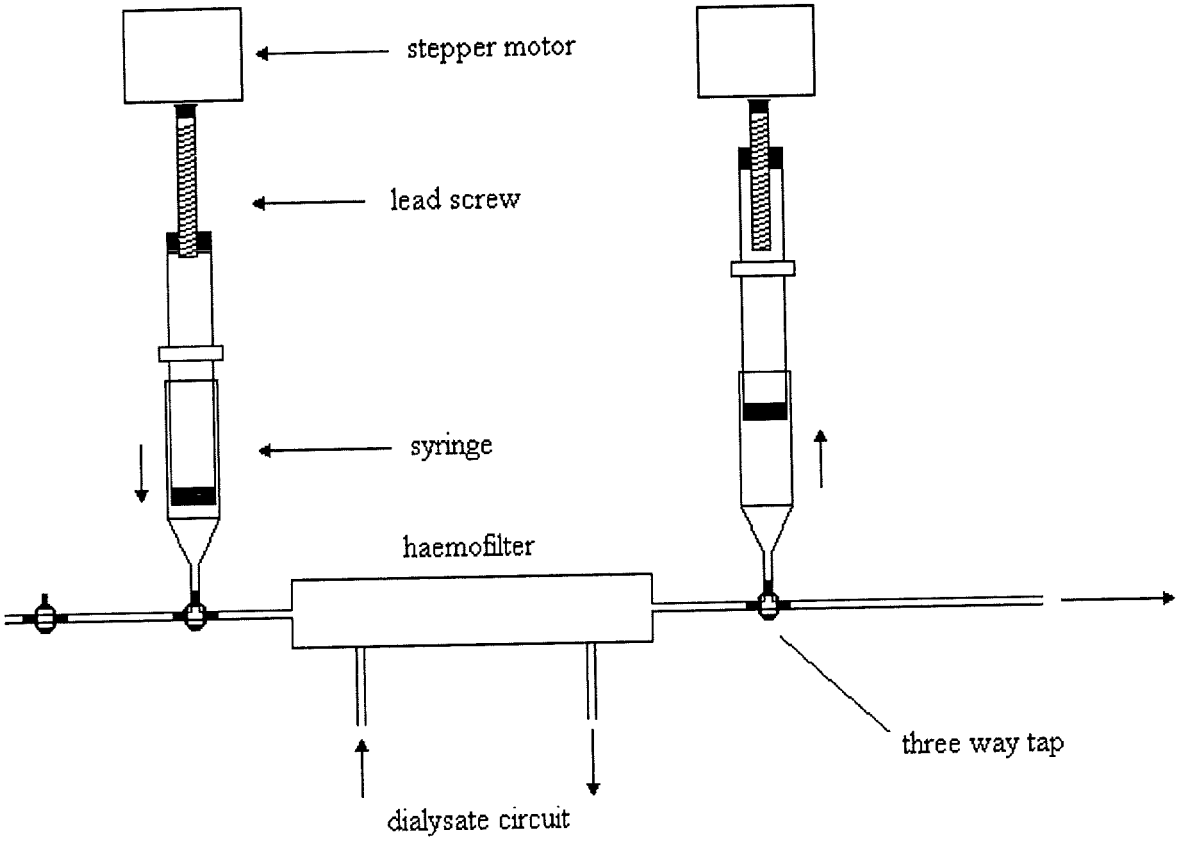


Figure 4.5 Two Way Circuit using Two Syringe Pumps

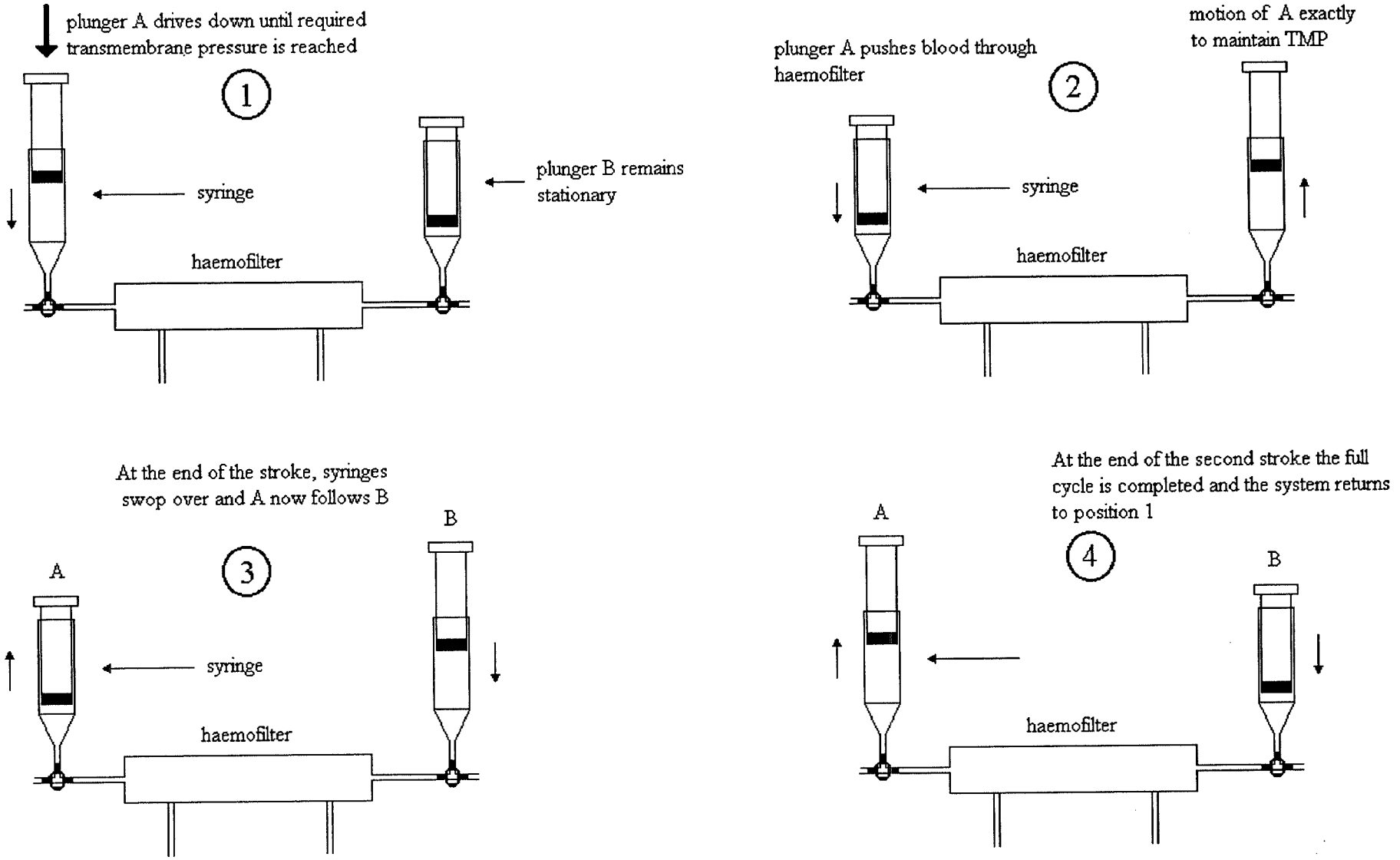


Figure 4.6 Maintenance of TMP in Secondary Cycle

CHAPTER 5. DESIGN OF MECHANICAL AND ELECTROMECHANICAL COMPONENTS

Introduction

This chapter addresses two main areas of design, the syringe drivers and the three way tap drivers. These are both very important parts of the system, and their design is considered in detail below.

5.1 Syringe Actuation

The first step in the design process for syringe actuation was to decide on the type of motor to be used. The choice for this is between a stepper motor, a conventional d.c. motor and a brushless d.c. motor. One potential advantage of a stepper motor is the possibility that a position feedback system could be dispensed with, so the control system could be less complex. This is discussed further in section 5.1.5. However, this type of design can lead to problems if the motor is expected to drive large loads - the motor can miss a step, so positional accuracy is lost. A d.c. motor with position sensing feedback does not suffer from this problem. A brushless d.c. motor has further advantages. It requires less maintenance, as there are no brushes to wear out. Its operation will be cleaner as no brush debris will be produced.

However, the stepper motor has the overwhelming advantage of requiring a simpler control system than the other two. Being in some respects a 'digital' device, it lends itself very well to computer control.

Having decided on the type of motor to be used, the next step was to calculate the specification needed from the motor/gearbox/lead screw combination. The requirements must be calculated for the 2 main phases of operation of the system.

5.1.1 *Filtration Phase Calculations*

Referring to the specification in chapter 3, the blood flow rate through the haemofilter is fixed at 10 ml/min under all operating conditions. The maximum required ultrafiltration rate is 15 ml/h, or 0.25 ml/min. Figure 5.1 (reproduced from

the data sheet for the Hospal Miniflow 10 haemofilter) can now be used to estimate the transmembrane pressure required. Extrapolating from this graph, a figure of 36 mm Hg is obtained. The data sheet also gives values for the blood pressure drop (ΔP) along the haemofilter. A figure of 61 mm Hg is quoted for a TMP of 50 mm Hg and a blood flow rate (Q_B) of 15 ml/min. As the variation in ΔP with TMP is very small it can be ignored for these purposes. The Pouseille formula for laminar flow in a pipe (see section 5.1.2) implies a linear relationship between ΔP and Q_B. Therefore, ΔP at a blood flow rate of 10 ml/min can be calculated:

$$\Delta P = 61 \times \frac{10}{15} \approx 41 \text{ mm Hg}$$

Transmembrane pressure is defined as follows:

$$TMP = \frac{P_1 + P_2}{2} + P_n \qquad (5.1)$$

Where P_1 and P_2 are the pressures at each end of the haemofilter, and P_n is the pressure applied to the ultrafiltration outlet. In this case P_n can be assumed to be zero. So, since $\Delta P = P_1 - P_2$, the values of P_1 and P_2 can be calculated.

P_1 = 15.5 mm Hg
P_2 = 56.5 mm Hg

So, the maximum pressure that the motor needs to be able to exert during the filtration phase is 56.5 mm Hg.

5.1.2 *Blood Withdrawal and Return Phase Calculations*

The design calculations were based on a typical catheter that might be used with the system. This is a Broviac 6.6 Fr Single Lumen Venous Catheter. The pressure needed to access the blood through this catheter can be roughly calculated using the Pouseille formula for laminar flow in a pipe:

$$\Delta P = \frac{128\mu LQ}{\pi d^4} \qquad (5.2)$$

where μ = viscosity of the fluid

L = length of pipe

Q = volume flow rate

d = diameter of pipe

It should be noted that the pressure varies as the fourth power of the diameter of the tube. This equation applies to Newtonian fluids, i.e. those fluids where the viscosity does not vary with shear rate. Blood is not a Newtonian fluid - it is shear thinning. This means that the viscosity goes down as the shear rate goes up. So a value calculated by the Pouseille formula should be an overestimate. For these calculations it is better to arrive at a figure that is too high rather than one that is too low. The variation in the viscosity of blood with shear rate is not that high, so the Pouseille formula will provide a useful first approximation to the pressure needed. Blood viscosity is a very important parameter in the pressure calculations. It has a large influence on the power needed from the stepper motors. In blood it is a fairly complex function of many different parameters - e.g. rate of shear, temperature and haematocrit. These calculations assume a constant dynamic viscosity of 5.46×10^{-3} Pa s. [86] This is the highest value quoted in this work, being the value measured at a shear rate of 1 s^{-1}.

The Broviac Catheter has an internal diameter of 1 mm. Assuming a 0.1 m length of tubing is placed inside the patient, the overall length of the catheter is 45 cm. The blood flow rate required is 10 ml/min (the same as in the filtration phase).

Therefore,

$\mu = 5.46 \times 10^{-3}$ Pa s

L = 0.45 m

Q = 10 ml/min $\approx$ 0.17 ml/s = 0.17×10^{-6} m^3/s

d = 1×10^{-3} m

So,

$$\Delta P = \frac{128\,\mu LQ}{\pi d^4} = \frac{128 \times 5.46 \times 10^{-3} \times 0.45 \times 0.17 \times 10^{-6}}{3.14 \times (1 \times 10^{-3})^4} = 1.7 \times 10^4 N/m^2$$

Converting to mm Hg, the pressure required is 130 mm Hg. This is over twice as much as that required during the filtration phase. So, the maximum pressure that needs to be generated inside the syringe by the linear actuator is approximately 130 mm Hg.

The force generated on the syringe plunger can now be calculated:

For a 10 ml syringe:

radius of plunger = 7.0 mm = 7.0×10^{-3} m

Pressure = 1.7×10^4 N/m^2

$$\Rightarrow \text{Force} = pA = p \times \pi r^2 = 2.62 \text{ N} \qquad (5.3)$$

The force needed to overcome friction between the plunger and the syringe must be added to this figure. This was found by experiment to be approximately 1 N for a 10 ml syringe. So the total force required is roughly 3.6 N. It must be remembered that these pressure calculations do not take into account any blockages in the vascular access line - extra force will be needed to overcome these blockages.

The speed of the plunger can also be calculated for a flow rate of 10 ml/min:

$$Q = \dot{x} \times \pi r^2 \qquad (5.4)$$

where Q = volume flow rate = 0.17×10^{-6} m^3/s

$\dot{x}$ = speed of plunger

r = radius of syringe = 7.0×10^{-3} m

$$\dot{x} = \frac{Q}{\pi r^2} = \frac{0.17 \times 10^{-6}}{3.14 \times (7.0 \times 10^{-3})^2} = 1.1 \times 10^{-3} m/s \qquad (5.5)$$

So the speed of the plunger is 1.1 mm/s.

The parameters for the lead screw system have thus been established. It must deliver at least a force of 3.6 N at a speed of 1.1 mm/s.

Formulae can be easily derived to relate force and linear velocity to torque and angular velocity in a lead screw system.

For angular velocity:

$$\dot{\theta} = \frac{2\pi\dot{x}}{L} \qquad (5.6)$$

where L = lead screw pitch (in mm)

$\dot{x}$ = linear velocity (in mm/s)

$\dot{\theta}$ = angular velocity (in rad/s)

$$\text{Also, Angular speed in rev / min} = \frac{60\dot{x}}{L} \qquad (5.7)$$

An energy method can be used to relate force and torque:

work done on piston of syringe = work done by torque on lead screw

$$\Rightarrow Fx = T\theta \qquad (5.8)$$

Considering a single revolution of the lead screw,

$\theta = 2\pi$

and x = L (pitch of lead screw)

$$\Rightarrow T = \frac{FL}{2\pi} \qquad (5.9)$$

Therefore a lead screw pitch of 1 mm would give a speed of

$$\frac{60 \times 1.1}{1} = 66 \text{ rev / min} \quad \text{for a linear velocity of 1.1 mm/s}$$

and the torque would be $\frac{3.6 \times 0.001}{2\pi} = 0.58$ mN m for a force of 3.6 N.

The efficiency of the gearbox and lead screw combination needs to be taken into account - once this has been done a final figure for the torque required from the motor can be arrived at.

5.1.3 *Motor, Gearbox and Lead Screw Selection*

The above calculations are based on the final specification that was reached for the prototype system. Selection of the linear actuator components was based on a different initial specification. The main difference was the internal diameter of the catheter - this was 0.5 mm, not 1 mm. The figures originally used were 42.1 N and 3.25 mm/s.

Initially the lead screw, motor and gearbox were considered separately. A selection of anti-backlash lead screw nuts were looked at. Positional accuracy is important in the syringe driver, so an anti-backlash nut seemed an obvious choice. After considering products from HPC drives and the PIC corporation, the choice was narrowed down to a PIC PK6000 type. This has a maximum load of 10 kg. A disadvantage of this product is its size (50 mm long). Several suitable motors and gearboxes were available. However, the cost of buying separate components and assembling them was considerably greater than the cost of a ready made unit.

The choice of ready made units was narrowed down to two - the Portescap P310 - L10 combination and the RS standard linear actuator. The Portescap product was in many ways preferable, but the delivery time was too long, so it was decided to use the RS product. This has a maximum starting force of 125 N, a top speed of 9.5 mm/s, and an increment of 0.025 mm. The price (£79) was also far better than anything else considered.

5.1.4 *Linear Actuator Mechanism*

A system was now needed that would hold the syringe rigidly in place and transmit the force from the actuator to the syringe plunger effectively, bearing in mind that sideways forces on the actuator lead screw were to be kept to a minimum. Figures 5.2 to 5.4 illustrate 3 possible designs.

The first diagram illustrates the use of two sets of bearings and guide rods,

and an end to end arrangement of the syringe and lead screw. This arrangement provides the minimum of lateral forces on the lead screw and stepper motor. However, this arrangement takes up a lot of space, increasing the overall size of the machine. Figures 5.3 and 5.4 show two possible arrangements where the lead screw, syringe and guide rod are all placed side by side, thus reducing the size of the mechanism. An analysis of the lateral forces in the mechanism shows that the arrangement of figure 5.4 produces the least strain on the stepper motor and lead screw. So this was the configuration that was chosen.

The final design is shown in figure 5.5. The basis of the design is a linear bearing running along a track rod. This minimises the lateral forces on the lead screw. The syringe plunger is gripped by a holding device which consists of three steel rods that slide over the top flange of the plunger. This device slides back and forth on a V shaped runner so that the syringe can be easily removed from the machine.

The stepper motor produces a substantial maximum force - 125 N. This is enough to cause serious damage to the mechanism if a control failure led to the mechanism being driven beyond its limits of travel. This problem was solved by placing microswitches at each of the limits of travel. The two switches were wired in series with the power supply to the stepper motors, in a normally closed configuration. So if the travel limits are exceeded in either direction the power to the stepper motor is cut off, preventing any mechanical damage. The switch at the bottom end of the plunger travel is positioned so that it does not affect the operation of the control microswitch.

5.1.5 *Positional Feedback*

The selection of the RS linear actuator limited the choice of transducers that could be used with the motor. Rotary encoders cannot be attached to this type of motor. So the choice was limited to linear encoders. Various linear sensors were considered, but all were found to be unsuitable. The plunger moves back and forth over a distance of approximately 6 cm - no commercially available sensor could be found that could operate over this length of travel.

The problem was solved by adopting a partially open loop approach to the positional control of the syringe driver mechanism. The control software counts each individual clock pulse that is sent to the stepper motor. So the control system can determine the plunger position to an accuracy of 0.025 mm. This of course

assumes that the stepper motor does not miss any steps. This problem is dealt with by placing a microswitch at the bottom end of the plunger travel. So if any steps are missed, the control software can reset itself each time the plunger reaches the bottom of the barrel and regain correct positional data. This 'intermittent' positional feedback was found to work very well in practise. By monitoring the number of clock pulses sent out and the microswitch states coming back the software can keep very reliable control of the stepper motors and deal with any possible failure mode. It also has the advantages of being cheap and easy to implement.

5.1.6 *Syringe Barrel Holders*

A mechanism was needed that would hold the syringe barrels firmly in place, but allow them to be removed quickly as needed. Initially this was made from plastic pipe fittings. 16 mm retaining clips were used to hold the barrel in place on either side of the barrel flange. A spring loaded retaining clip was also made from a piece of pipe fitting. The pipe fittings were mounted on an aluminium block to provide the correct height for the syringe relative to the linear actuator mechanism.

Although this design was sufficient for the testing stage of the prototype, its 'home made' appearance was considered unsuitable for clinical use. The holder was later redesigned and made entirely from aluminium (see chapter 7).

5.2 Three Way Tap Drivers

5.2.1 *Actuation*

Originally it was proposed to do away with the 3 way taps altogether, and use solenoid tube clamps instead to control the flow of blood. However, suitable solenoids were not readily available, so it was decided to pursue direct driving of the existing 3 way taps.

Several makes of 3 way tap were available, any of which could be used in the system. The Connecta TH type was chosen (see fig 5.6), as it had the smallest plastic to plastic bearing surface area, minimising the torque required to turn it.

The original specification called for the driver to be able to position the tap in

3 discrete positions spaced at 90° intervals, to an accuracy of a few degrees. Given this requirement, a DC servo motor was the obvious choice of actuator. Rotary solenoids were briefly considered as well, but found to be not feasible, as a suitable type of device was not commercially available. A reduction gearbox with a fairly high ratio was also an obvious choice. This was for two reasons. The first was that there was no need for fast movement between tap positions. The second was that a high reduction ratio reduces the problems associated with overshoot of the target position.

Servo motors of the type used in radio control models provided a suitable ready made motor and gearbox combination. They were found to produce sufficient torque to easily turn the 3 way tap (Futaba model S3001 was the type initially used). The PWM circuitry inside the servo motor was bypassed - the motor was connected directly to the external control circuitry.

5.2.2. *Detection*

Microswitches were considered for providing positional feedback. They have the advantage of simplicity, but the disadvantage of lack of reliability. It was decided to use solid state detectors, namely infrared transmitters and photodetectors. A device was chosen that has an infrared diode and a phototransistor mounted in the same plastic moulding, with a 3 mm gap between the devices (see fig 5.7). This device can detect a slot cut in the edge of an aluminium disc that is rotating in the gap between the diode and the transistor.

The initial design for the position detection system is shown in figure 5.8. The photodetectors are positioned 90° apart, and there are two slots cut in the edge of the aluminium disc, also 90° apart. This arrangement allows 3 discrete positions to be detected, at 90° intervals. However, when the control programming was done, it was found that the system was very unreliable. The problem was that the detectors do not change state simultaneously when both slots are rotating underneath them (position 2 in figure 5.8). The detector state at position 2 could be momentarily the same as that at positions 1 or 3. So a simple control program cannot be used to provide reliable control.

The problem was solved by changing the specification. It was discovered that 3 distinct tap positions were not in fact needed. 2 would be sufficient with no loss of functionality in the blood flow control. The number of slots in the disc was

reduced to one, so that 2 positions 90° apart could be reliably detected.

5.2.3 *Connection between Driver and Tap*

It was important that the connection was inflexible but also easy to engage and disengage. This is so that the clinical apparatus can be easily attached and removed, as required by the specification. This was achieved using a 3 mm thick steel disc. The exact shape of the turning handles on the 3 way tap (see figure 5.6) were cut into the disc using a CAM wire eroder machine. The handles of the 3 way tap could then be push fitted into the steel disc.

5.2.4 *Initial Design of System*

The initial design for the tap driver system is shown in figure 5.9. The tap driving disc is mounted on top of the optical sensor disc. The photodetectors are mounted on matrix board.

5.2.5 *Three Way Tap Retainers*

The clinical apparatus is not particularly rigid, and it would be fairly easy for the 3 way taps to work loose from the driving discs during operation. This had to be avoided at all costs, as this would be extremely unsafe. A device was needed to hold the taps securely in place during operation, but also that would still allow the clinical apparatus to be easily removed. The design is shown in figure 5.10. It consists of a spring loaded aluminium block that can be tightened down onto another base block. The shape of the body of the 3 way tap is milled into the underside of the block, so that it fits neatly over the tap. The tap body is also prevented from rotating, thus increasing the accuracy of the tap target positions and reducing the mechanical strain on the clinical apparatus.

Summary

The requirements of the syringe drivers were analysed for both the filtration and blood withdrawal/return phases of the operating cycle. This allowed a suitable linear actuator to be selected. The mechanism for driving the syringes was then

designed.

The specification for the 3 way tap drivers was more straightforward. This was based on a measurement of the torque required to turn the 3 way tap, the need for accuracy in positional control, and the physical connection between driver and tap.

The design of the electronics that provide the interface between the electromechanical components and the computer system is the subject of the next chapter.

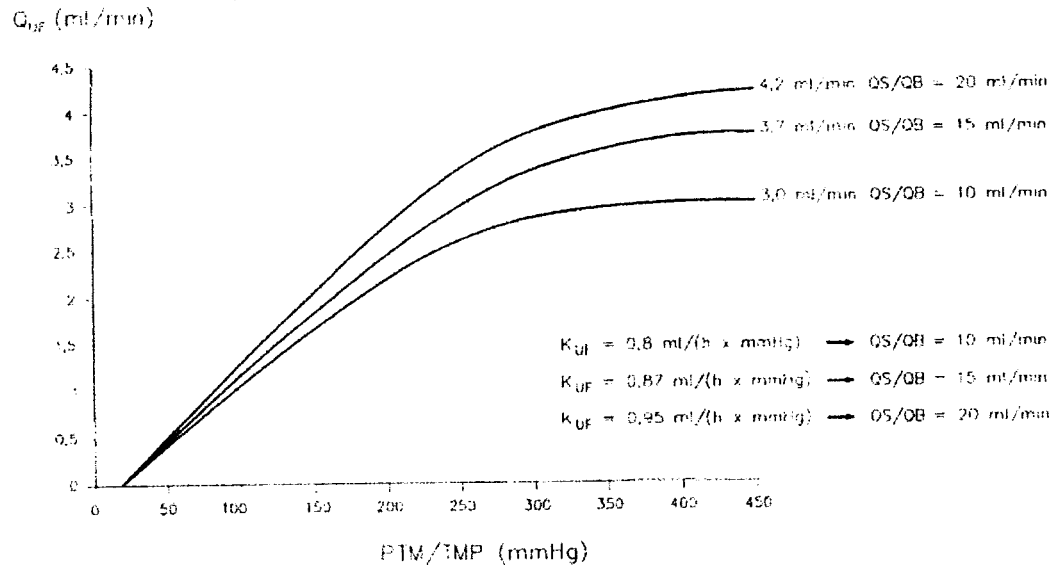


Figure 5.1 Graph of Ultrafiltration Rate vs. Transmembrane Pressure [87]

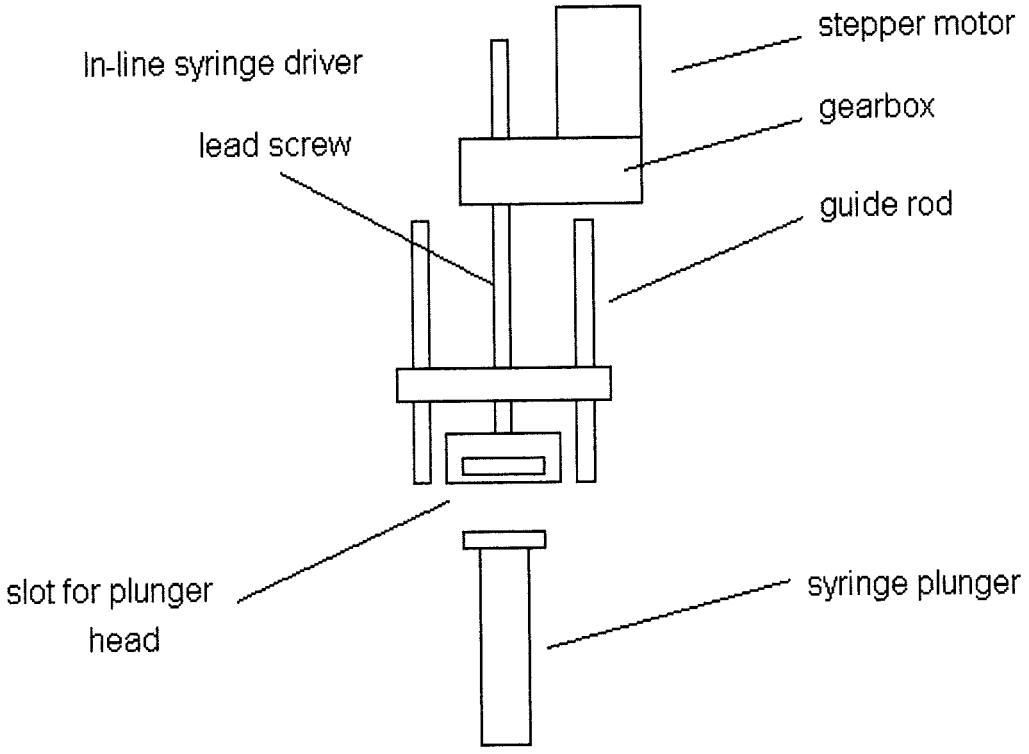


Figure 5.2 Linear Actuator Design 1

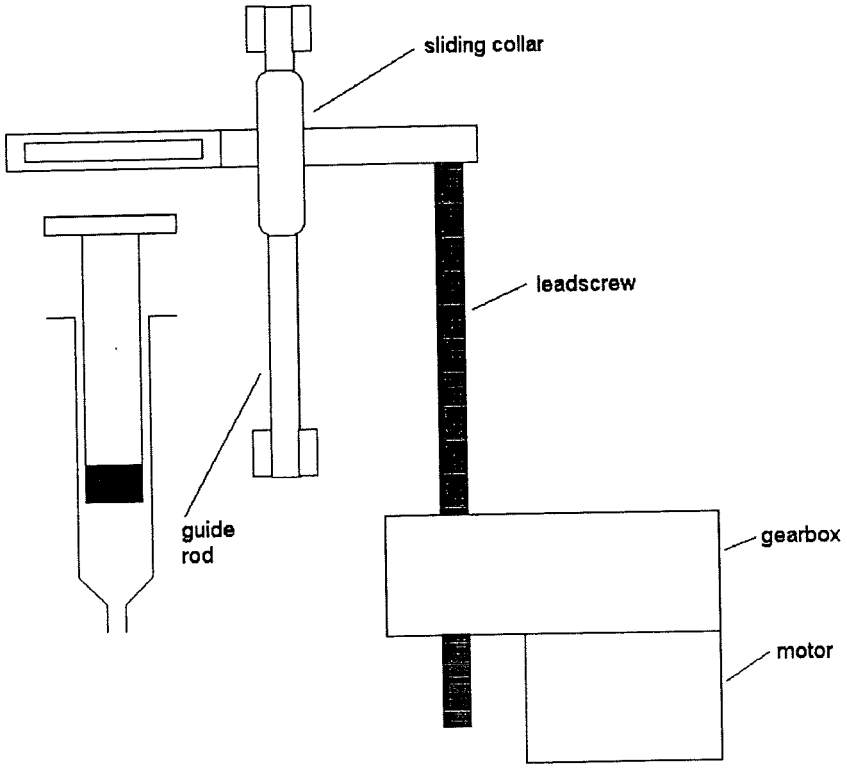


Figure 5.3 Linear Actuator Design 2

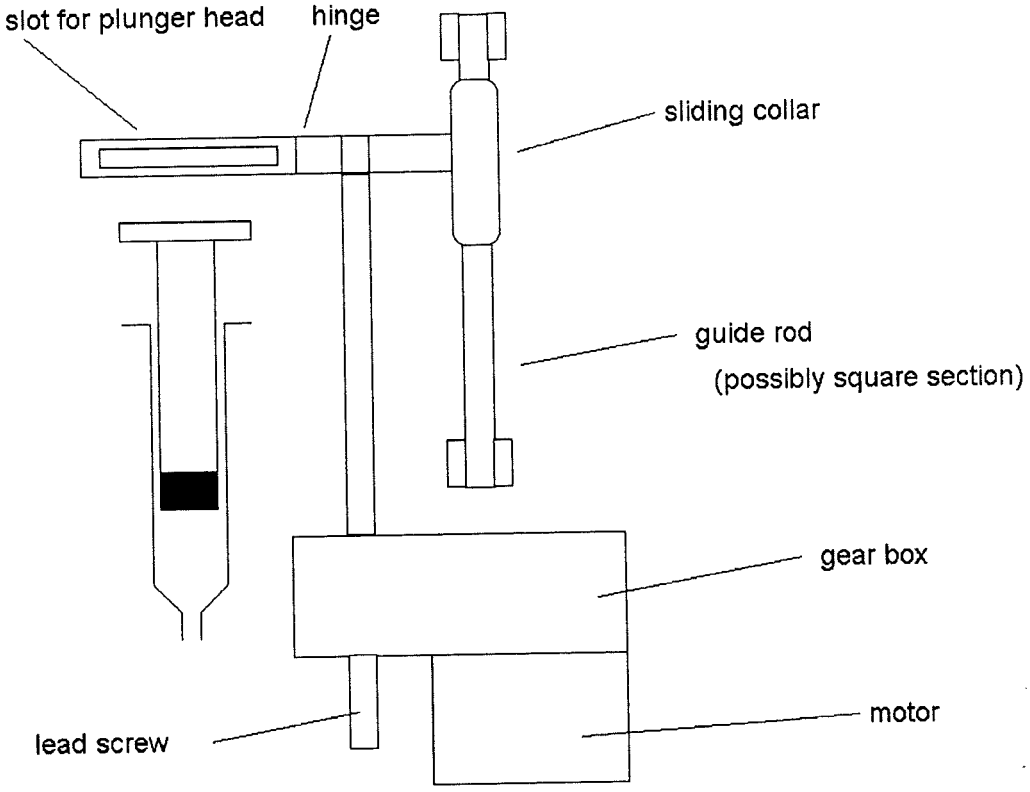


Figure 5.4 Linear Actuator Design 3

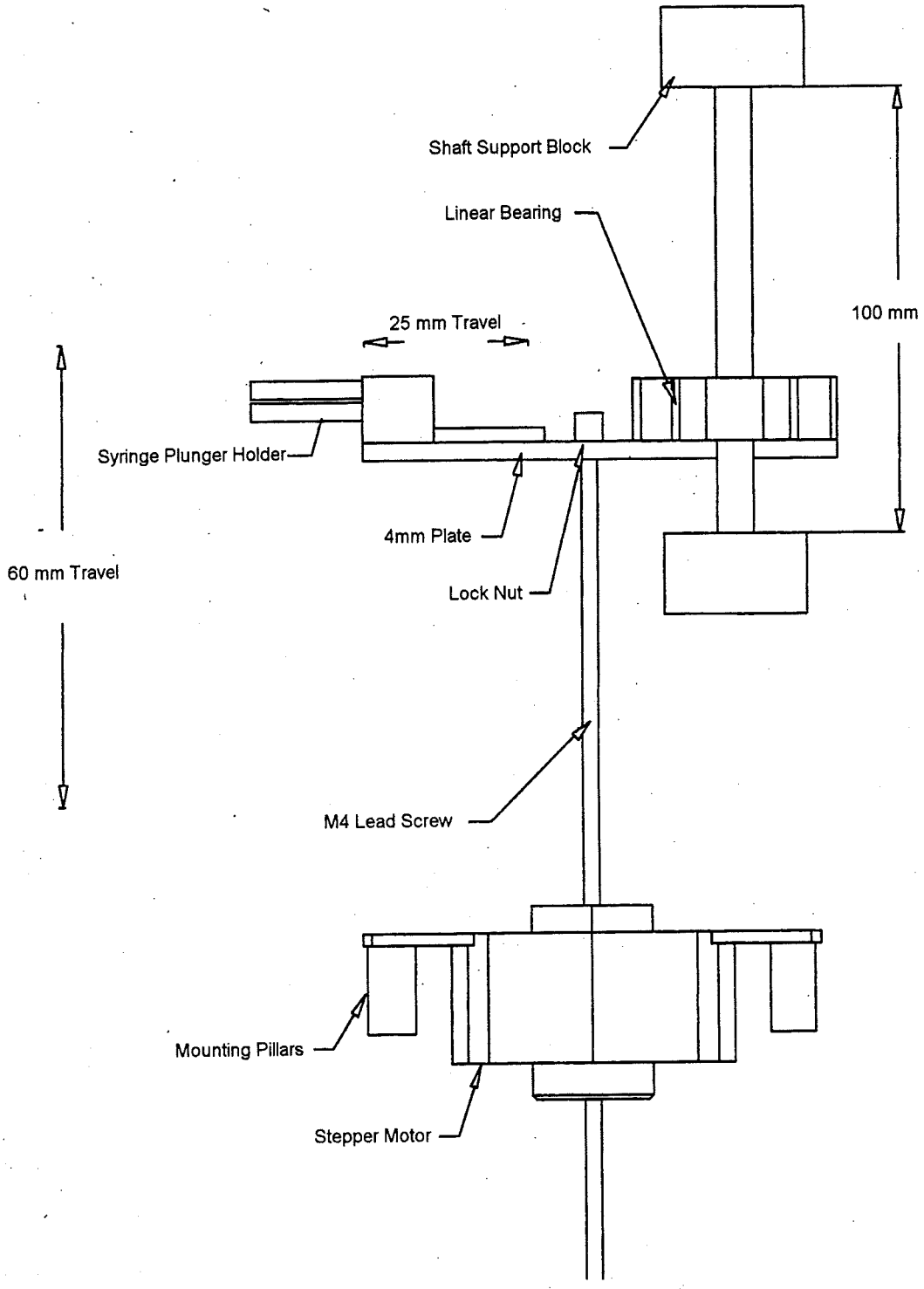


Figure 5.5 Syringe Driving System

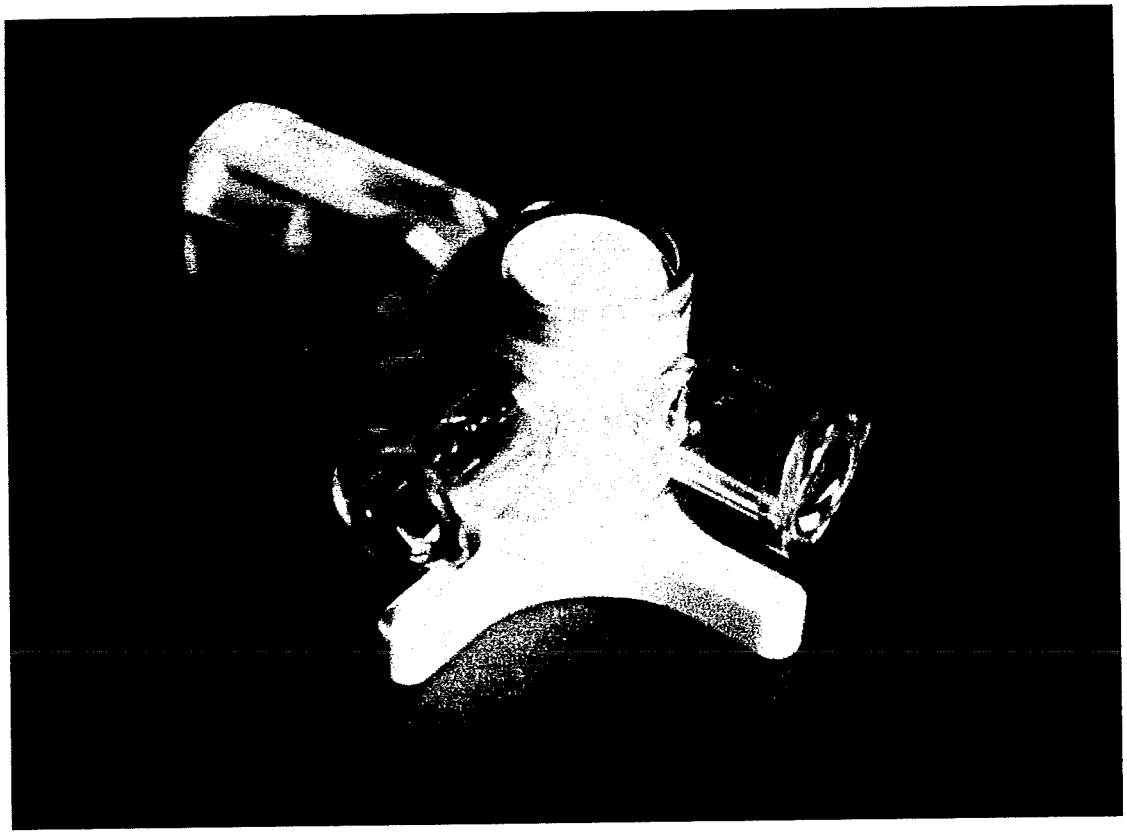


Figure 5.6 Connecta 3 Way Tap

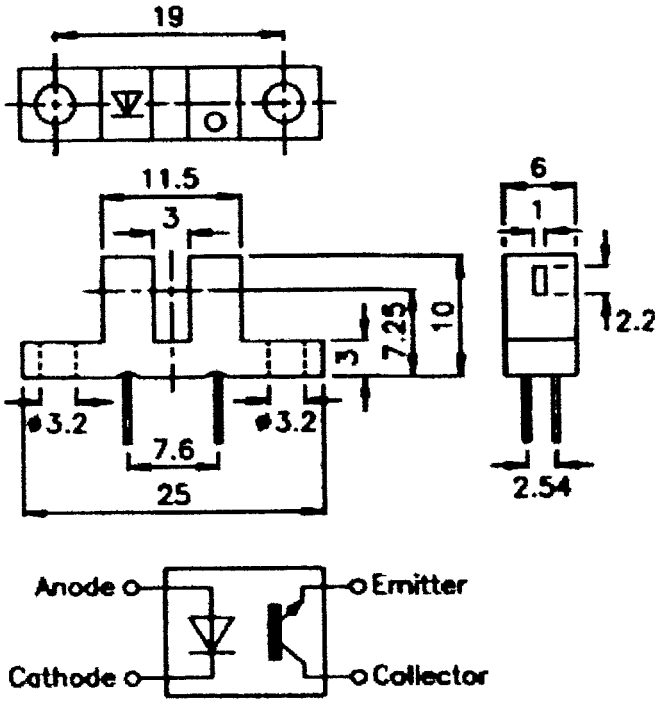


Figure 5.7 Photodetector Packaging [88]

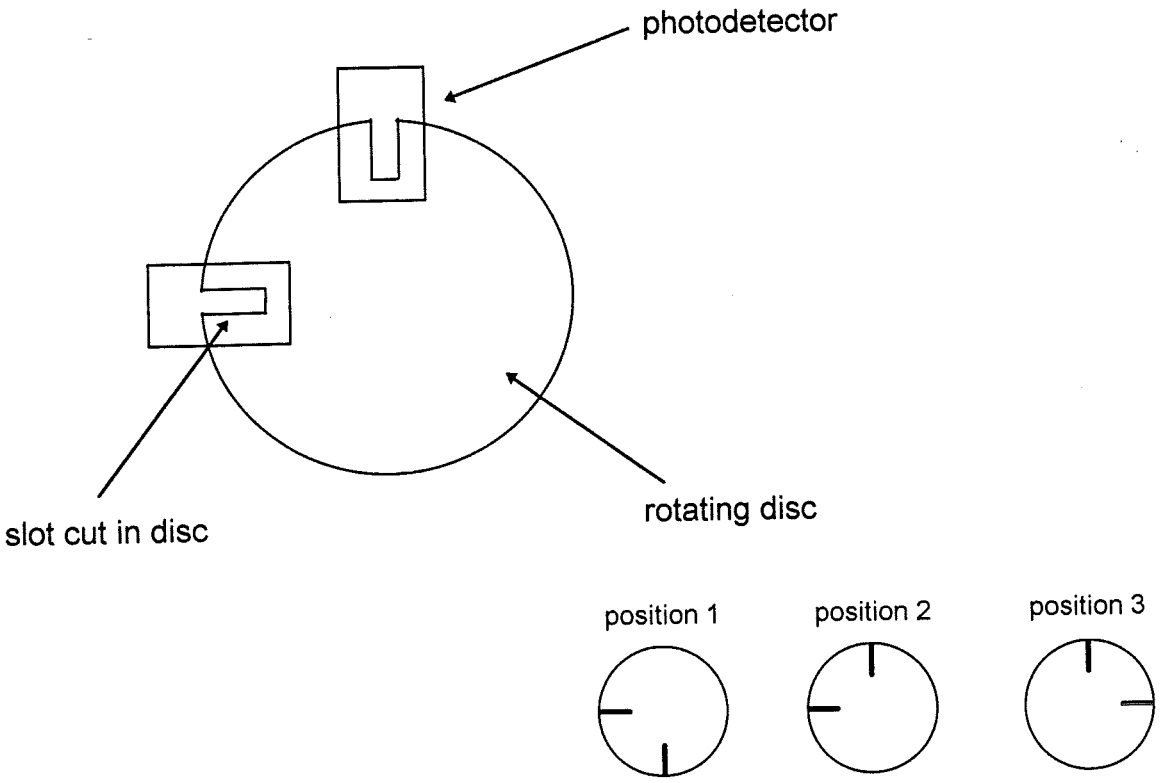


Figure 5.8 Tap Driver Photodetectors

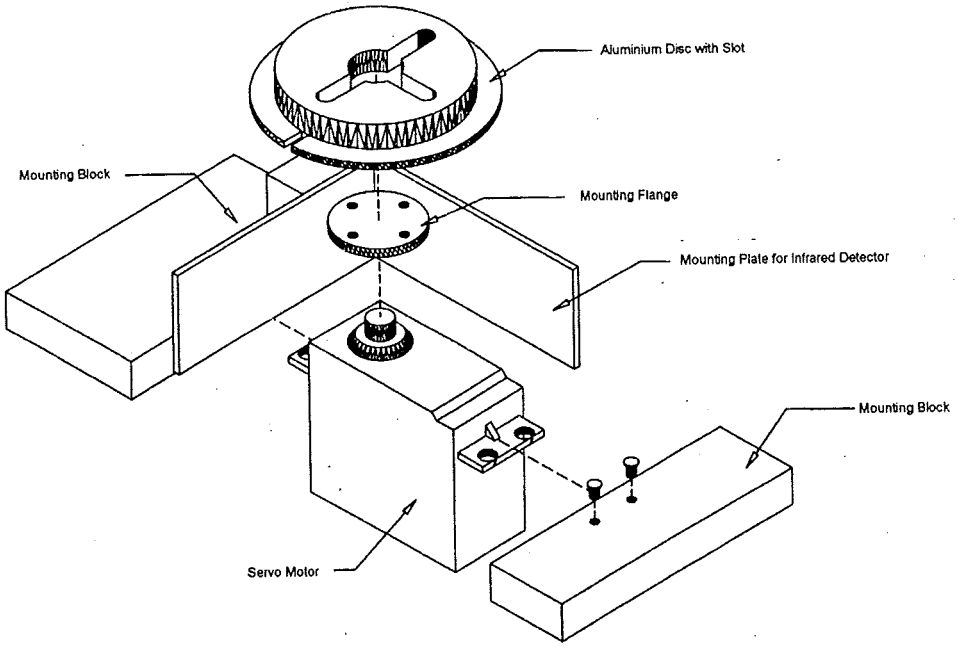


Figure 5.9 Original 3 Way Tap Driver

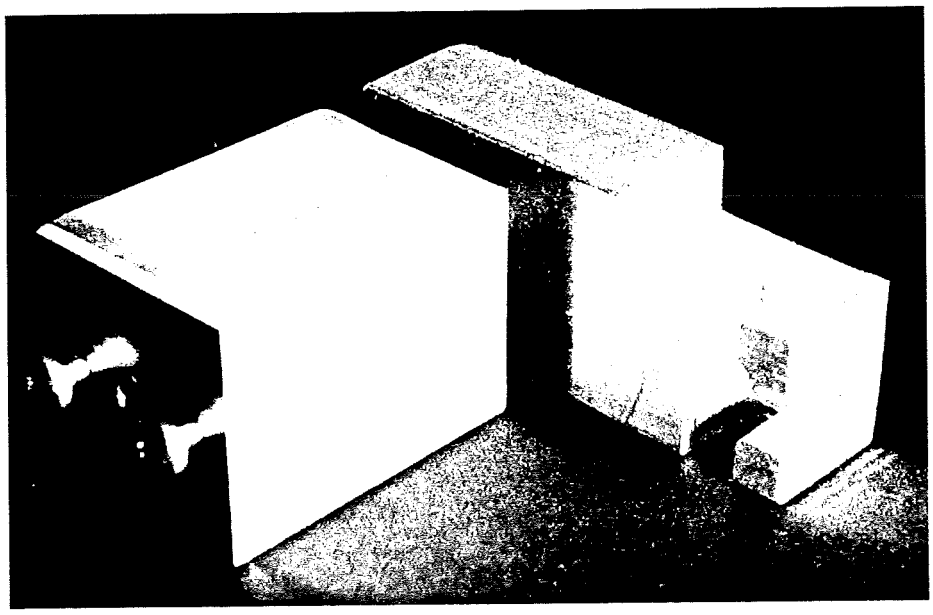


Figure 5.10 Three Way Tap Retainer

CHAPTER 6. DESIGN OF THE INTERFACE ELECTRONICS

Introduction

The electronic circuitry needed to interface the various sensors and transducers to the control system is the subject of this chapter. The power supply is also dealt with in section 6.5.

6.1 Syringe Drivers

6.1.1 *Stepper Motor Drive Boards*

The torque required from the stepper motors is quite high. However, the speed needed is quite low. In chapter 5 it was shown that the linear velocity needed from the actuator is 1.1 mm/s. Each step is 0.025 mm long so this speed corresponds to 44 steps/s.

There is no need for good dynamic performance (rapid acceleration or deceleration). In view of all this it was decided that sophisticated (and therefore expensive) drive electronics were unnecessary. An RS unipolar drive board (stock no. 217-3611) was chosen. There was no need for controller or oscillator circuitry as the clock pulses were to be produced in the software.

6.1.2 *Stepper Motor Series Resistors*

The dynamic performance of a stepper motor being driven from this type of board is greatly increased by the addition of resistance in series with the motor coils. This is because series resistance reduces the time constant (L/R) of the coil circuit, thereby reducing the current rise time and increasing the available torque. The series resistance needed can be calculated using this ohm's law formula:

$$R = \frac{V_S - V_M}{I_M} \qquad (6.1)$$

where V_S = Supply voltage

V_M = Rated motor winding voltage

I_M = Rated motor winding current

The motor and drive board were to be run from a 15 V supply. The rated voltage was 12 V and the resistance per phase 25 Ω. This gives a current per phase of 0.48 A.

So,

$$R = \frac{15 - 12}{0.48} = 6.25\ \Omega$$

And the power dissipated in each resistance = 3 V X 0.48 A = 1.44 W.

The 6.25 Ω resistance was made up of 4 smaller preferred value high power resistors in series. Figure 6.1 shows how the resistors are connected to the motor coils.

6.1.3 *5 V - 12 V Conversion Circuitry*

The clock and direction inputs to the stepper motor drive board require a 12 V logic signal. The outputs from the data acquisition board use the normal TTL 5 V system. It was necessary to provide some sort of interface between these two systems. A simple transistor switch was used. The circuit diagram is shown in figure 6.2. The base bias resistor was chosen as the highest value that would still saturate the transistor, to minimize the load on the DAQ board output. R_2 was chosen to provide the optimum voltage swing between the on and off states. The circuit acts as an inverter - a 5 V input signal produces a 0 V output, and a 0 V input produces a 12 V output. 4 such circuits were needed, to drive the clock and direction inputs on the two stepper motor drive boards.

6.1.4 *Position Feedback Microswitch Circuits*

At the initial design stage it was thought necessary to provide some signal conditioning for the syringe driver microswitches. A standard schmitt trigger circuit was used, shown in figure 6.3. This circuit eliminates the effect of contact bounce in the microswitch, providing a smooth transition between logic levels. However, the algorithm that was eventually used (see chapter 8) to control the movement of the syringe plungers made this circuit redundant. It is included here for completeness.

6.1.5 *Power Cut Off Microswitches*

These were discussed in chapter 5. Figure 6.1 shows how the microswitches were wired into the power supply lead from the stepper motor drive board to the motor itself.

6.2 Three Way Tap Drivers

6.2.1 *Motor Drive Circuit*

It would be possible to drive the 3 way taps with a motor that only turned in one direction, since there are no limit stops on the taps. The taps must turn between 2 positions 90° apart. So movement between the 2 positions would involve either a 90° movement or a 270° movement, depending on the starting position. The speed of rotation is fairly slow because of the high ratio of the gearbox, so a 270° movement represents several seconds' delay in obtaining the target position. While this is not of critical importance, it does represent a slight reduction in the efficiency of the machine. Balanced against this is the much simpler circuit that can be used to drive the motor. A high power transistor switch can be used, the transistor being turned on and off by the TTL logic signal from the DAQ card connected to the base.

Eventually it was decided to use bidirectional control. This was partly for the reasons outlined above, and partly because the servos that were used had limit stops built into the casing.

Several different circuit ideas were considered. The first is illustrated in figure 6.4. The 2 inputs from the DAQ card would carry the logic codes for the desired servo positions, e.g. 10 for one position and 01 for the other. Positional feedback would then be provided by a combinational logic circuit which would send out the appropriate on/off and direction signals to the motor. Such a circuit would take control of position away from software and put it into hardware. A Karnaugh map was drawn up to see what logic would be needed to implement this circuit. This proved to be quite complex, making the idea impractical.

The decision was made to control position through the computer program, to make the circuitry simpler. This separates the actuator and sensor parts of the system.

Another idea is shown in figure 6.5. Here a TTL signal switches the motor on and off via TR2. A direction signal is applied to TR1, which switches the relay. The relay controls a set of double pole, double throw contacts, which reverse the direction of the voltage applied to the motor. This circuit has the advantage of simplicity, but the disadvantage of reduced reliability. This is because the relay introduces a mechanical element to the system.

A more reliable circuit is shown in figure 6.6. Here 4 power transistors are connected in a bridge arrangement. When TR1 and TR4 are switched on and TR2 and TR3 are switched off, current will flow through the motor from left to right. When TR2 and TR3 are switched on, the current will flow from right to left, providing directional control.

An even better circuit is shown in figure 6.7, based on the same idea as the previous circuit. This is the circuit that was chosen for the application. It gives anticlockwise and clockwise movement at constant speed under the control of the two digital signal lines A and B. The L272M is a power operational amplifier which is capable of driving a small DC motor directly, having a maximum output current of 1A. The circuit works on an open loop principle, and since there is no feedback the gain of the amplifiers is very high. The inverting inputs of both op amps are held at a constant 2.5 V by the potential dividers. A logic 1 (5 V) signal at the non inverting input will result in the output going fully positive (i.e. to the supply voltage). A logic 0 at the same input will make the output go to 0 V. So if the inputs are both at the same logic level, the motor will be switched off. If input A goes high, and input B goes low, then 5 V will appear on the left hand side of the motor, and 0 V on the right hand side, so the motor will run. If the logic levels are reversed, i.e. input A goes low and input B goes high, the direction of the voltage across the motor will be reversed, so it will run in the opposite direction. The diodes protect the outputs of the op amps from any back e.m.f. effects generated by the motor.

6.2.2 *Infrared Detectors*

The interface circuits for the infrared detectors were straightforward to design. The circuit diagrams are shown in figure 6.8. Again the output from the detector is a TTL compatible logic signal. The circuit has an inverting action. The output is low when the infrared beam is shining on the phototransistor and high when the beam is interrupted. The series resistor for the infrared diodes was calculated as follows. Each diode passes a current of 60 mA when forward biased

and the voltage drop across it is 1.5 V. With a 5 V supply the voltage drop across the resistor is therefore 3.5 V and the resistor value is given by:

$$R = \frac{V}{I} = \frac{3.5}{60 \times 10^{-3}} \approx 58\ \Omega \qquad (6.2)$$

The nearest preferred value to this is 66 Ω. Combining all 4 diodes in parallel requires a resistance of 66/4 = 16.5 Ω. The total current is 240 mA, so the power requirement is:

$$P = I^2R = (0.240)^2 \times 16.5 = 0.95\ W \qquad (6.3)$$

This requirement was met by using 2 33Ω 0.5 W resistors in parallel.

The value of the series resistor for the phototransistor was found by experiment. A value of 3.3 kΩ was found to give the best combination of voltage output levels for logic 0 and logic 1.

6.3 Pressure Transducer Amplifier

A standard clinical pressure transducer was selected for pressure sensing inside the apparatus - an Abbott Transpac IV monitoring kit. This is luer terminated so it can be easily incorporated into the existing apparatus. It has an operating pressure range of -50 to 300 mm Hg, which is more than sufficient for this application. It is a piezoresistive device, connected as part of a bridge, so there are 4 electrical connections to the transducer. The output voltage is very small, 5 μV/V/mm Hg, so some amplification of the signal is needed before it can be fed to the analogue input of the DAQ card. This is provided by the differential amplifier circuit shown in figure 6.9. This configuration is particularly suitable for a bridge transducer because the circuit amplifies the difference between the two voltages, V_a and V_b.

A precision instrumentation operational amplifier type OP184 was chosen for the circuit. This is one of the few precision devices that can operate from a single voltage power supply. This is a big advantage, as it simplifies the overall power supply requirements of the machine.

The gain of the circuit had to be determined with reference to the input

requirements of the DAQ card. The analogue inputs can be configured to be bipolar (reading both positive and negative voltages) or unipolar (reading just positive voltages). With a single voltage power supply, clearly a unipolar configuration is needed. The gain of the analogue input can be adjusted in software to produce different signal input ranges. The maximum range is 0 - 10 V and the minimum is 0 - 100 mV. The actual range chosen is not that important, as the resolution is the same for all ranges. However, the range must obviously match well with the output range from the differential amplifier.

Simple circuit theory gives a formula for the gain of the differential amplifier circuit:

$$V_{out} = \frac{R_2}{R_1}(V_a - V_b) \qquad (6.4)$$

With a 5 V power supply, the output from the transducer is $5 \times 5 = 25$ μV/mm Hg. In chapter 5, the pressure rise on blood return was calculated to be approximately 128 mm Hg, and this was the maximum pressure expected to be generated in the extracorporeal circuit. A similar drop is expected on blood withdrawal, so the pressure range over which measurements must be taken is $2 \times 128 = 256$ mm Hg. This is equivalent to a change in output from the transducer of 256×25 μV = 6.4 mV. A gain of 100 was chosen to give an output range of 640 mV from the amplifier. This fits quite well into the 0 -1000 mV range obtained with a gain setting of 10 on the DAQ board. Using the formula above, a gain of 100 can be achieved by making R_2 = 100 kΩ and R_1 = 1 kΩ.

While utilising as much as possible of the DAQ board input range, to obtain maximum resolution, it is very important that the output signal does not exceed the DAQ board range. Problems arose in this respect during testing. These will be dealt with in chapter 7.

Both pressure rises and pressure falls must be measured. Since the circuit is being driven from a single rail power supply, ambient pressure must produce an output that is in the middle of the range of possible output voltages. This was achieved by attaching a potentiometer to the offset null pins of the operational amplifier as shown in figure 6.9. With this arrangement, V_a -V_b is set to zero, and the potentiometer is adjusted until V_{out} is mid-range.

6.4 DAQ Board Interface

The connections between the interface electronics and the data acquisition board are shown in figure 6.10. It was decided that it would be useful to have a visual indication of the states of the digital lines. This is useful for monitoring of system function and also to speed up fault finding. The circuit shown in figure 6.11 was designed for this purpose. Each digital line is fed into a TTL inverting buffer, provided by a 74LS04 integrated circuit. This buffer limits the current drain from the DAQ board. The output of the buffer drives a light emitting diode. The circuit provides a 'double inverting' action, so when the LED is on, the line is at state 1. 14 of these circuits were required.

6.5 Power Supply

Once all the electronics had been designed, constructed and tested, the power supply requirements could be calculated. The stepper motors run on a 15 V supply, as do their drive boards. The McClellan servo motors that were used in the final design require a 12 V supply. The 5 V/12 V converter circuits and the inputs to the stepper motor drive boards run at 12 V. Everything else uses a 5 V supply. The total power requirements at each voltage were calculated, and the calculation was confirmed using the Farnell bench power supply. The requirements were:

5 V at 1 A
12 V at 0.5 A
15 V at 2 A

The total requirements were reduced by the fact that the stepper motors and the tap drivers are not required to run at the same time. A suitable triple output supply could not be found, so it was decided to use a dual output supply instead, and obtain the 12 V supply by using a DC-DC converter. A switched mode unit was preferred to a conventional linear one, as they are more efficient.

The unit chosen was an Astec LPT63. This was the cheapest power supply that could deliver the current needed at both 5 V and 15 V. Its specifications are 7 A at 5 V and 2.8 A at 15 V.

A DC-DC converter is necessary for the 12 V supply because of the current needed. Cheaper voltage regulator chips would not be able to supply such high

currents. A Computer Products type BXA15-12S12-F was chosen. This can supply 1.25 A at 12 V from a supply ranging from 9 V to 18 V.

One disadvantage of switched mode power supplies is that they require a minimum loading to function correctly. Without this loading the output voltages are very unpredictable. It was necessary to add shunt resistors to both the 15 V and the 5 V lines to achieve these minimum loads. The calculations for this follow.

With all systems connected to the Farnell power supply, the minimum current drawn by the system at any time was measured. For the 5 V line, this was 0.28 A. The minimum load needed by the power supply is 0.7 A. So the current needed in the shunt resistance, I_s, is given by :

$I_s = 0.7 - 0.28 = 0.42$ A

$$\text{So, } R_s = \frac{V}{I_s} = \frac{5}{0.42} = 11.9\ \Omega \qquad (6.5)$$

This can be made up from the preferred values 6.8 Ω and 4.7 Ω in series.
The power dissipated in each resistor is as follows:

For the 6.8 Ω resistor,

$$P = I^2R = (0.42)^2 \times 6.8 = 1.2\ \text{W} \qquad (6.6)$$

For the 4.7 Ω resistor

$$P = (0.42)^2 \times 4.7 = 0.83\ \text{W}$$

3 watt ceramic wirewound resistors were used.
For the 15 V line, the minimum current flows when neither stepper motors nor 3 way tap drivers are running. This was measured and found to be 0.23 A.
The minimum load needed by the power supply is 0.3 A.
So,

$I_s = 0.3 - 0.23 = 0.07$ A

$$\Rightarrow R_s = \frac{V}{I_s} = \frac{15}{0.07} = 214\ \Omega \qquad (6.7)$$

This was made up of the preferred values of 180 Ω and 33 Ω. A check of the power dissipated gives:

For the 180 Ω resistor:

$$P = (0.07)^2 \times 180 = 0.88\ W$$

For the 33 Ω resistor:

$$P = (0.07)^2 \times 33 = 0.16\ W$$

Again, 3 W ceramic wirewound resistors were used.
A schematic diagram of the power supply system is shown in figure 6.12.

Summary

The design of the electronics fell into five major areas. The syringe driver circuitry involved both actuation and position sensing, as did the circuitry for the 3 way tap drivers. The pressure transducer amplifier provides an interface between the pressure sensor and the analogue input of the data acquisition board. The DAQ board interface circuit provides a general interface between the DAQ board and the other digital interface electronics. The last area is that of the power supply. The requirements here were complicated by the need for three different supply voltages.

This chapter has dealt with the last major area of design. The next chapter deals with the construction and testing of the prototype system, as well as the design modifications that were necessary.

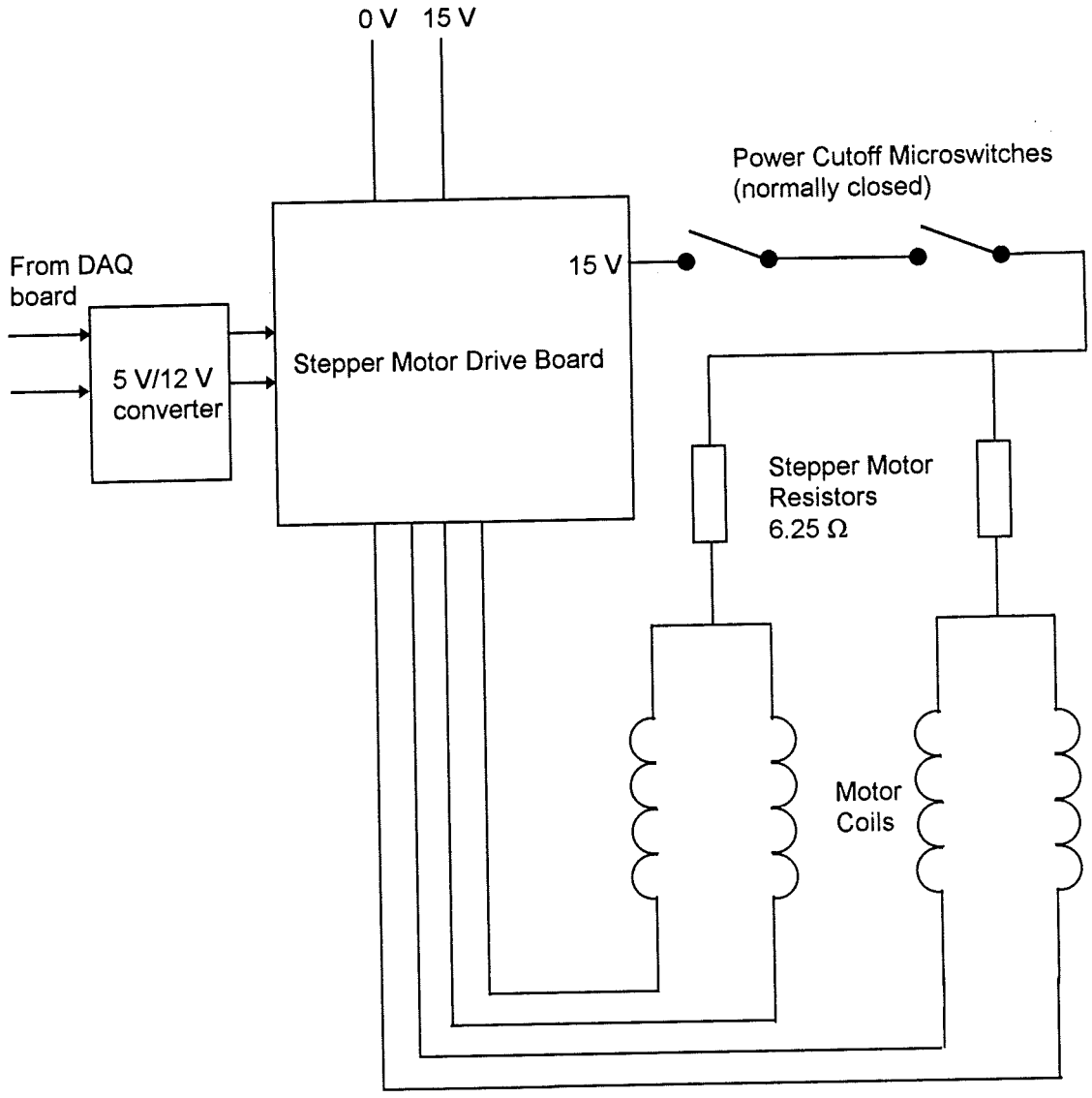


Figure 6.1 Stepper Motor Wiring

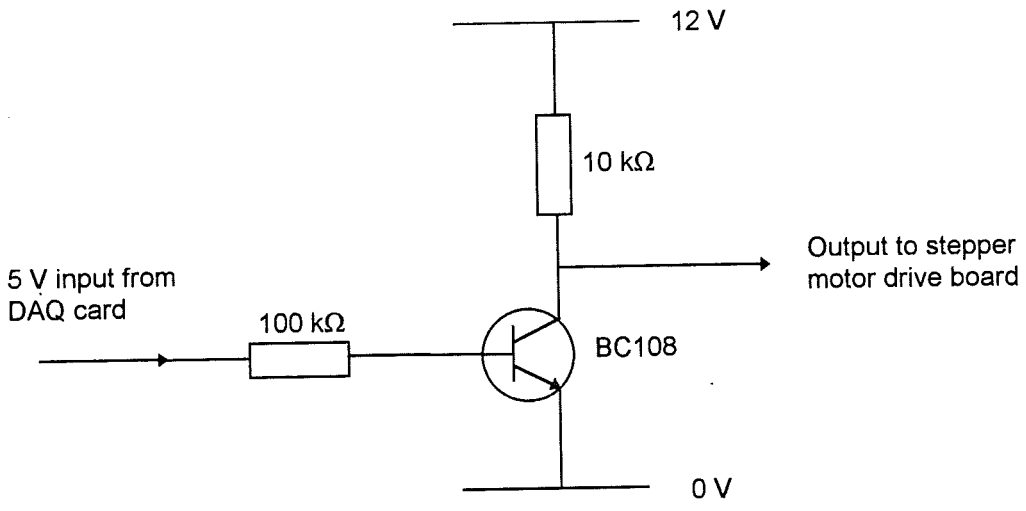


Figure 6.2 5 V / 12 V Converter Circuit

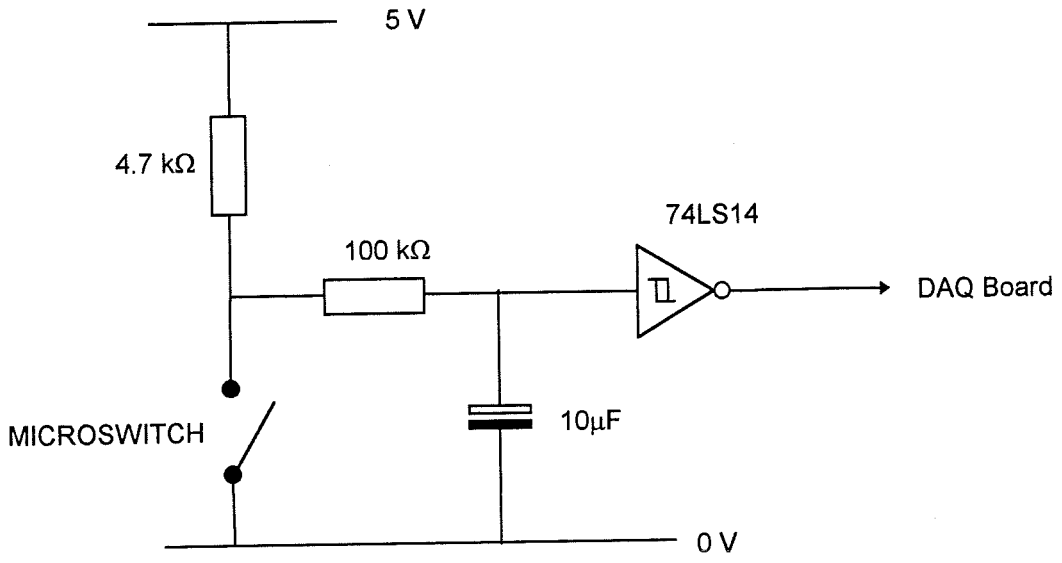


Figure 6.3 Microswitch Interface Circuit

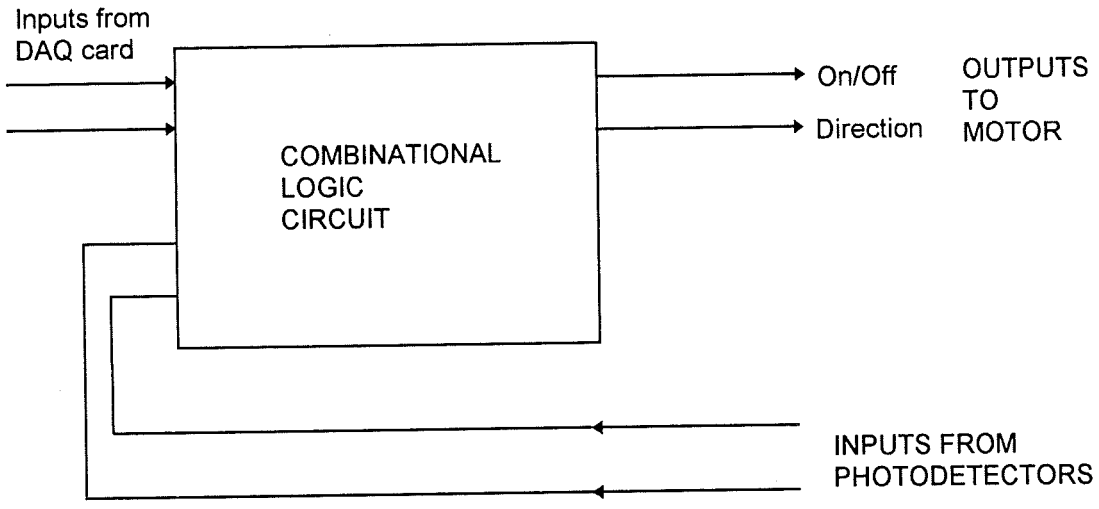


Figure 6.4 Motor Drive Circuit Design 1

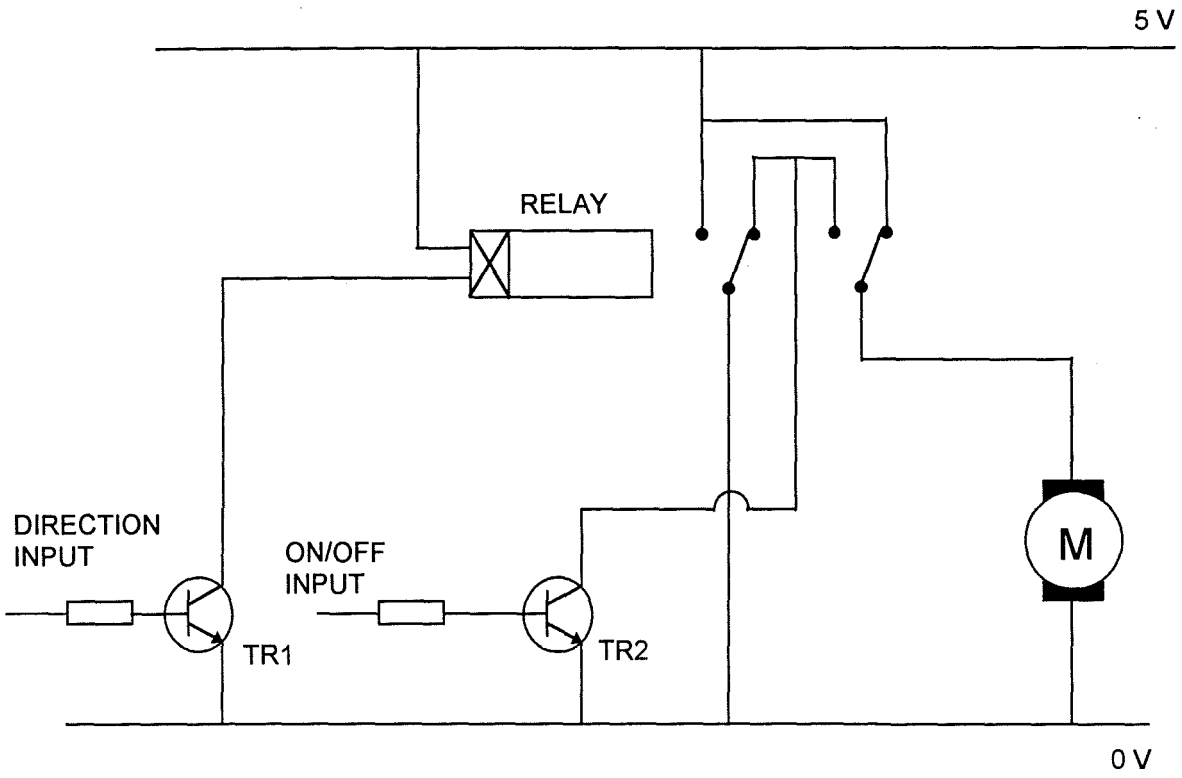


Figure 6.5 Motor Drive Circuit Design 2

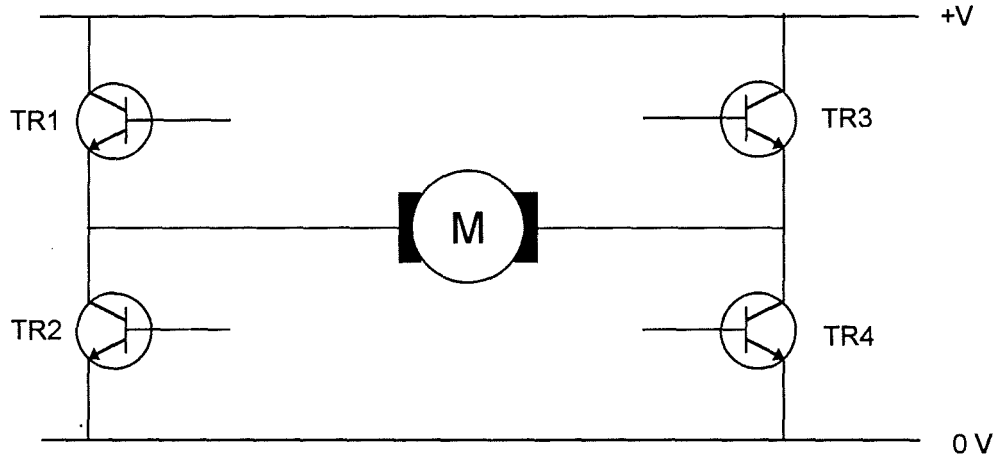


Figure 6.6 Motor Drive Circuit Design 3

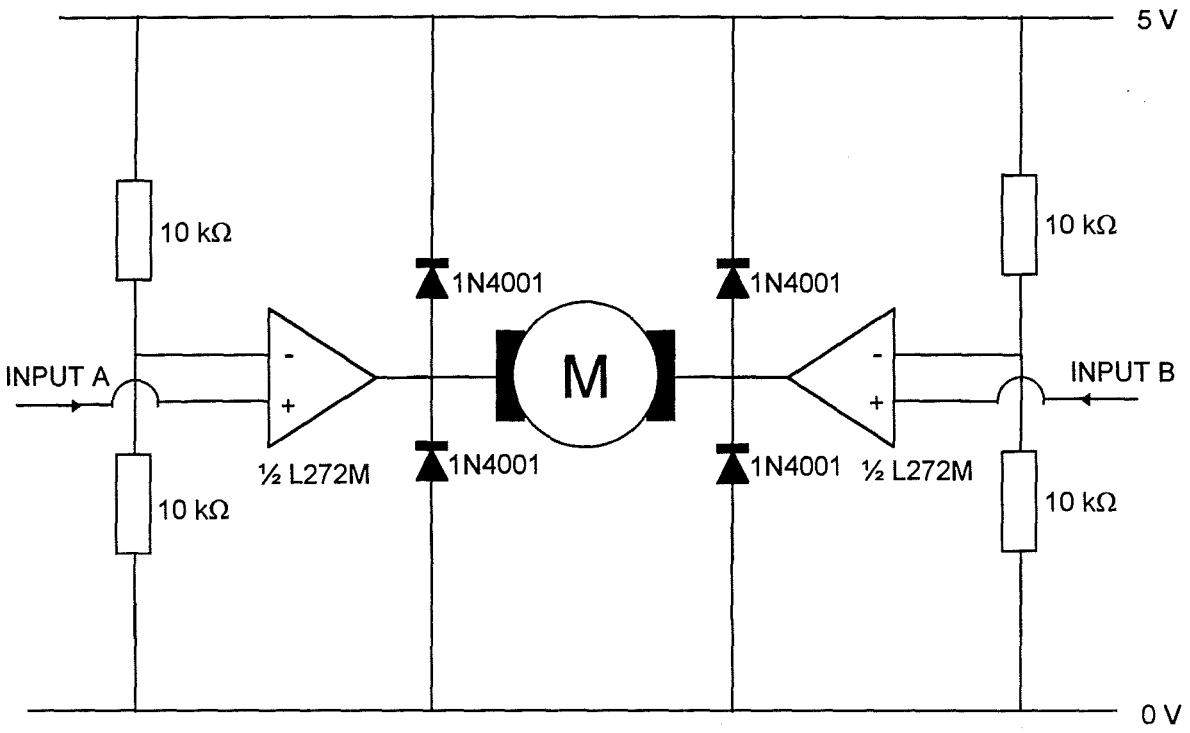


Figure 6.7 Final Motor Drive Circuit Design

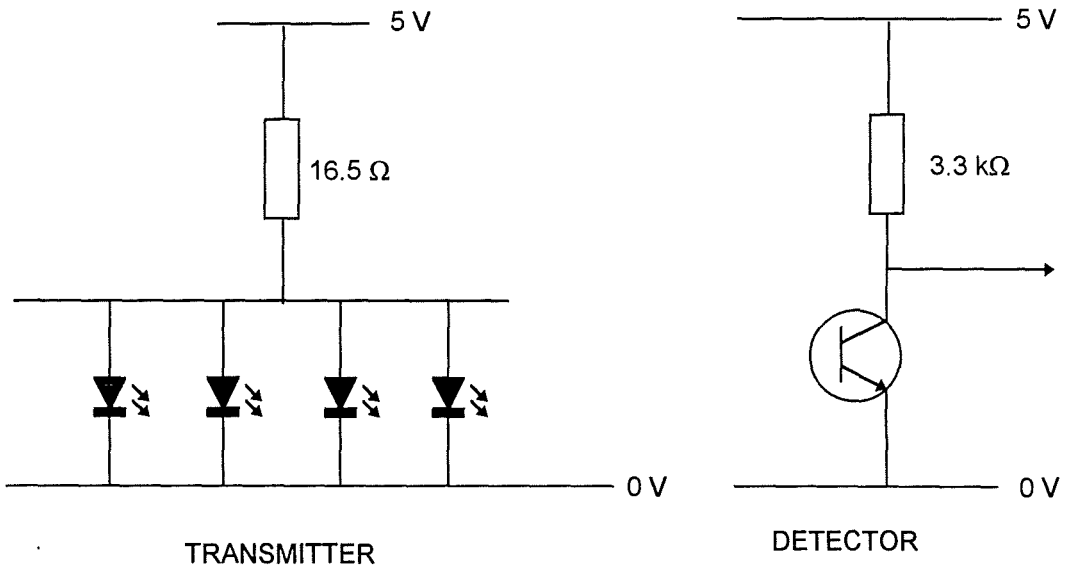


Figure 6.8 Infrared Transmitter and Detector

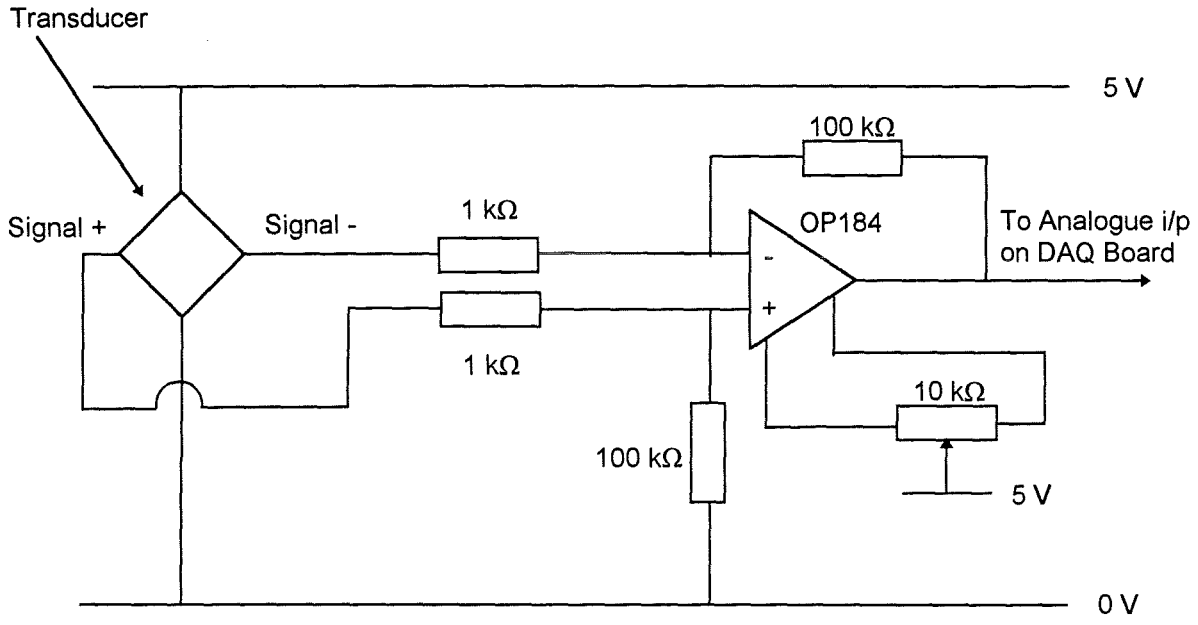


Figure 6.9 Pressure Transducer Differential Amplifier

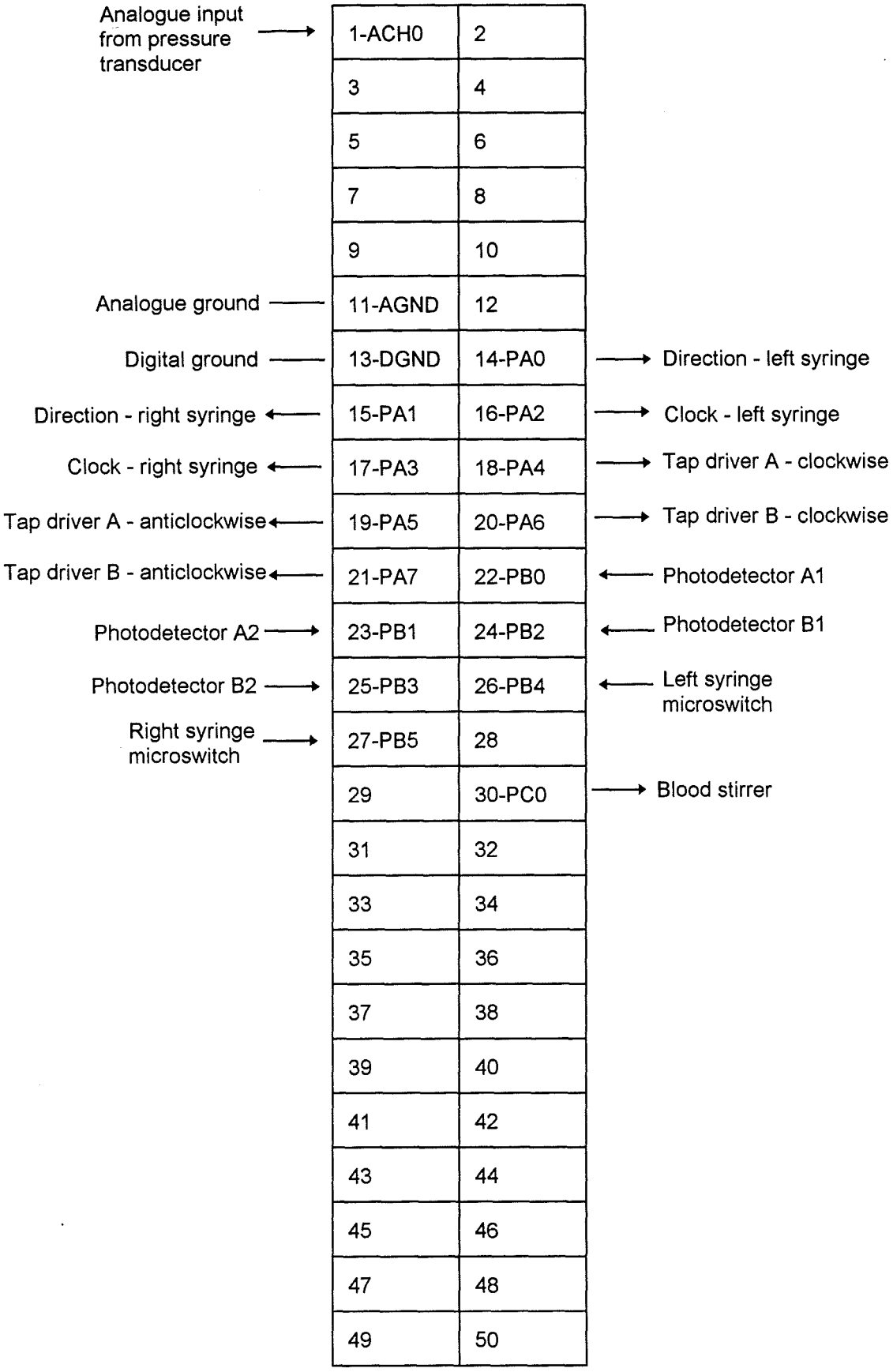


Figure 6.10 DAQ Board Connections

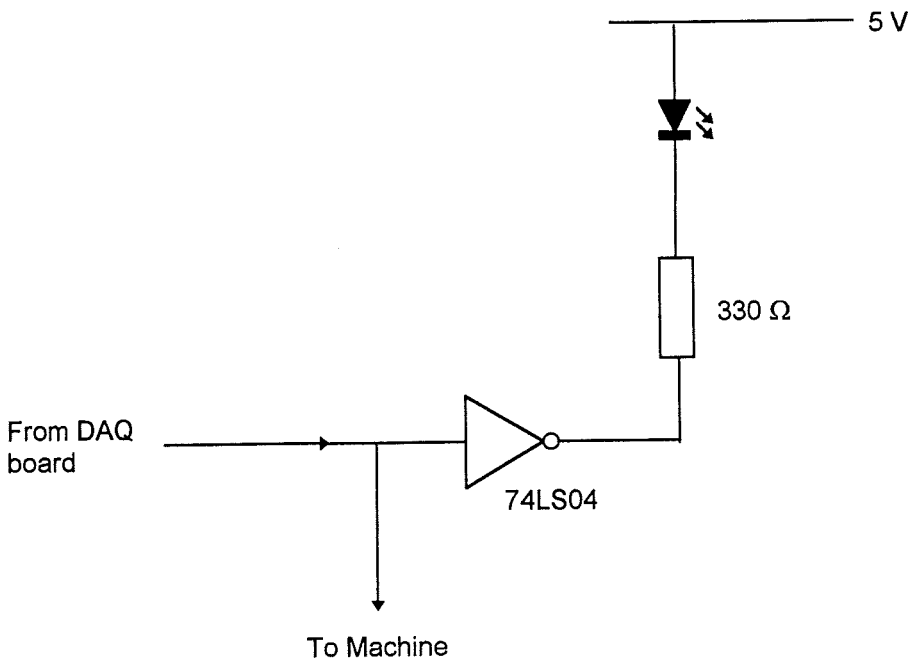


Figure 6.11 LED Indicator Board

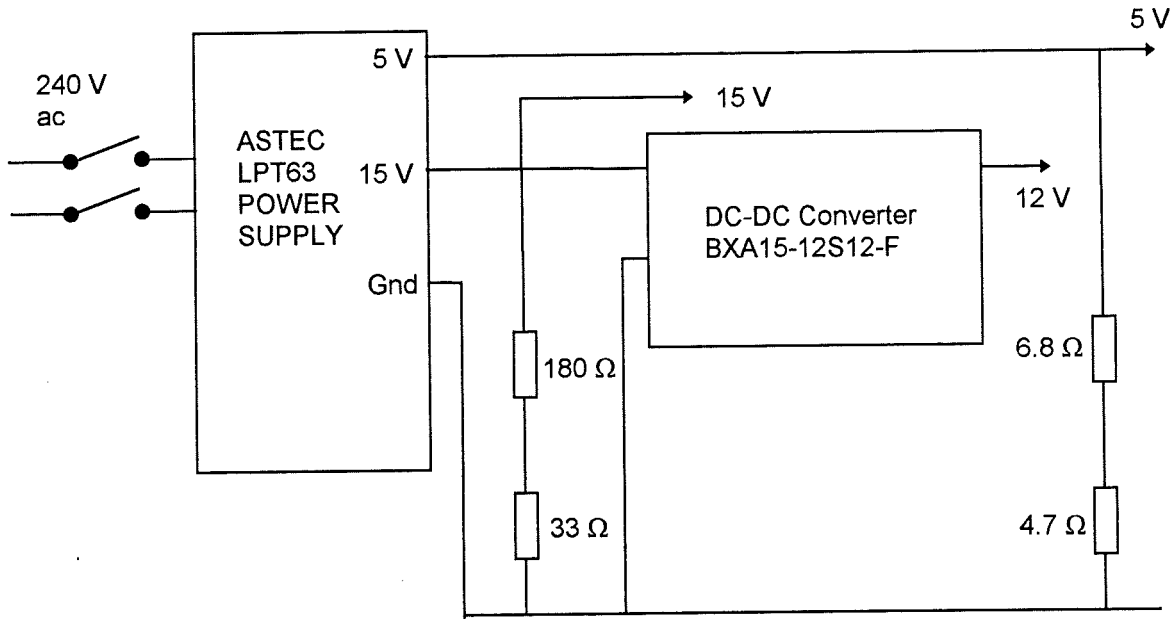


Figure 6.12 Schematic Diagram of Power Supply System

CHAPTER 7. CONSTRUCTION, TESTING AND DESIGN MODIFICATIONS

Introduction

The construction and testing of the prototype system revealed the need for several design modifications. This phase of the project is discussed below.

7.1 Construction and Testing of System Elements

Once the initial design work was complete, the construction and testing of the prototype was begun. This was split up into separate subsystems. Each was built and tested separately before being integrated into the system as a whole.

The syringe driver mechanisms were mounted on two 4 mm thick aluminium base plates for testing. The haemofiltration apparatus was held in place using plastic pipe fittings. The tap driver mechanisms were built onto a separate aluminium framework which was then attached to the base plates.

A prototyping plugboard was used to test the electronic circuits. These were powered with a Farnell dual bench power supply. Once the circuits had been successfully tested they were transferred to matrix board. The plugboard was then used to provide a temporary interface between the computer and the machine. This interface allowed the machine's systems to be controlled both automatically and manually, that is by the computer and by hand. This was particularly important while the computer software was being developed. The interface itself consisted of microswitches and logic gates that allowed dual control of each system.

7.2 Design Modifications

7.2.1 *Syringe Drivers*

The initial design for the syringe driver mechanism worked fairly well in initial testing. The V shaped slider mechanism did not work very well, for 2 reasons. Firstly, both sliding surfaces were aluminium, so they did not slide easily over each other. This problem could be solved by making one surface brass. A more serious problem was that the plunger gripper was too loose a fit on its track.

This is important because it would lead to inaccuracies in volumetric measurement during the filtration phase. The problem was solved by adding a hand tightening bolt which could be screwed down when the gripper was in place, holding the two sliding surfaces securely together.

The control and safety microswitches at the bottom end of the syringe plunger travel were mounted side by side. The existing layout did not contain enough space for both switches to be triggered by the connecting plate as it moved down. So an aluminium extension plate was added.

An even more serious problem was encountered when the machine was first tested using blood. After several hours of operation, there was a very large increase in the frictional force between the plunger and the inside of the syringe barrel. This was caused by something being deposited out of the blood onto the inside of the barrel. This increase in friction caused the rubber bung to gradually peel off the end of the syringe plunger during the upward movement of the plunger. Eventually the seal between plunger and barrel was lost and blood leaked by. The problem was solved by squirting a small amount of normal saline into the top of each syringe barrel, which provided lubrication between the 2 surfaces. This prevented the inside of the barrel from drying and so stopped the build up of friction. In clinical operation any fluid that was placed inside the top of the syringe barrels would have to be sterile.

7.2.2 *Three Way Tap Drivers*

The original design for the 3 way tap drivers worked well when the testing fluid was water. Unfortunately a similar problem occurred in the taps as did with the syringes, when the testing fluid was blood. Figure 5.6 shows the construction of a 3 way tap. The white plastic barrel inside rotates inside the clear plastic cylinder. There is a large plastic to plastic bearing surface area. When blood flows through the tap problems occur. The blood seeps into the bearing and dries out, creating a very large increase in the friction between the 2 surfaces, resulting in a higher torque being needed to turn the tap. This torque was measured using a simple lever and weight method, and was found to be approximately 3.8 kg cm.

A more powerful Futaba servo, model S9204, was tried in an effort to solve this problem. Although it had a specified torque of 9.5 kg cm, this servo was still not capable of consistently driving the taps once they had become contaminated with blood.

It was decided to solve the problem by using a much more powerful motor and gearbox combination. More calculations were done to determine the requirements of the motor and gearbox. A suitable source of motors and gearboxes was found from McLennan Servo Supplies Ltd.

The torque required to turn the tap had been determined as 0.4 N m. This was multiplied by a factor of 5 to give a very large safety margin, so the design figure was 2 N m. A reasonable speed of movement was fixed at 4 seconds for a ¼ turn, which is 3.75 rev/min. Assuming a typical no load speed of 3000 rev/min for a small DC motor, the maximum allowable gearbox ratio can be calculated:

$$i \leq \frac{3000}{3.75} = 800$$

The 120 series of gearboxes were the cheapest available that had sufficient rated torque, at 3 N m. The highest available ratio (500:1) was chosen to minimise the motor requirements. These were then calculated:
The motor torque is given by

$$M_i = \frac{M_o}{i \times \eta} = \frac{2}{500 \times 0.5} = 8 \text{ mN m} \qquad (7.1)$$

where M_o=Output torque = 2 N m

i = gearbox ratio = 500

η =gearbox efficiency = 0.5 (assumed to be 50 %)

The motor speed is given by

$$n_i = n_o \times i = 3.75 \times 500 = 1875 \text{ rev/min} \qquad (7.2)$$

where n_o= output speed

Motor type 16 111 was found to be the smallest motor that would satisfy these requirements, having a rated torque of 15 mN m at 2500 rev/min.

The voltage required to produce this torque and speed can be found from the following formula:

$$V = \frac{M_i R}{K} + K\omega \qquad (7.3)$$

where M_i = motor torque = 8×10^{-3} N m

R = rotor resistance = 6.2 Ω

K = torque constant = 0.035 N m/A

ω = angular velocity = $2\pi/60 \times n_i$ = 196 rad/s

So,

$$V = \frac{8 \times 10^{-3} \times 6.2}{0.035} + (0.035 \times 196) = 8.3 \text{ V}$$

Which is well below the 12 V available for this motor.

Finally, the current required to produce this torque can be calculated, to check that the power supply is capable of delivering it.

$$I = \frac{M_i}{K} = \frac{8 \times 10^{-3}}{0.035} = 0.23 \text{ A} \qquad (7.4)$$

This is well within the 1.25 A that the DC - DC converter can supply, and also below the 2.8 A that the power supply can provide.

To accommodate the new gearbox and motor, a new mounting and drive system was designed. The photodetector mountings were also redesigned. Figure 7.1 shows details of the new system. The new design also allowed the entire system to be mounted inside the machine casing. During testing of the initial design it was found that infrared emissions from artificial light (such as that from filament bulbs) could interfere with the infrared detectors. So it was important that they should be mounted inside the casing.

Finally the drive electronics had to be altered to accommodate the new voltage needed for the motor. The existing operational amplifiers could be used at the higher voltage. All that was necessary was to change the potential divider resistors so that 2.5 V was still supplied to the inverting inputs of the amplifiers. Referring to figure 7.2 the calculation is as follows:

$$\frac{V_1}{R_1} = \frac{V_2}{R_2}$$

$$\Rightarrow \frac{R_2}{R_1} = \frac{2.5}{9.5} = \frac{5}{19} \qquad (7.5)$$

If a value for R_1 of 10 kΩ is chosen to keep the current small, then

$R_2 = 2.63$ kΩ

Using preferred values, this can be approximated by two 4.7 kΩ resistors in parallel in series with a 330 Ω resistor (gives 2.68 kΩ). The operation of the circuit does not require the reference voltage to be exactly 2.5 V.

7.2.3 *Pressure Sensing System*

During the early stages of testing with blood, a problem arose in the system - an increased range of pressures became apparent in the circuit during the filtration phase. This was caused by the partial coagulation of blood within the fibres of the haemofilter. This in turn was caused by too little heparin being added to the blood before the testing was begun. The amounts needed to prevent coagulation inside the circuit are relatively high, at 5 units/ml of blood.

The pressure changes became so great that the system was no longer able to measure them. The output from the transducer amplifier reached either supply or ground voltage. This is worth looking at in more detail.

As was mentioned in chapter 6, the output from the pressure transducer is 25 μV/mm Hg. The gain of the amplifier is 100, so the output from the amplifier is 2.5 mV/mm Hg. The gain of the DAQ card is set to read input voltages in the range 0 - 5 V. The null offset potentiometer is adjusted to give an output of 2.5 V at ambient pressure, so that both pressure rises and falls can be measured. So, the maximum pressure change that can be measured in this configuration is

$$\frac{2.5\ \text{V}}{2.5\ \text{mV}} = 1000\ \text{mm Hg}$$

either above or below ambient pressure.

The problem was overcome by the addition of a multipole switch and extra gain resistors, so that the gain of the amplifier was adjustable - there are three

settings: 51, 75, 100. This gives more flexibility during the development and testing process. With a gain of 75, the measurable pressure change becomes 1333 mm Hg, and with a gain of 51 it is 1960 mm Hg. Figure 7.3 is a partial circuit diagram showing the changes that were made. In clinical use, any pressure change above 500 mm Hg is considered unsafe due to the red cell haemolysis that it can induce. Therefore, once the problems of heparinisation had been solved these extra pressure ranges were no longer needed.

7.2.4 *Syringe Barrel Holders*

As was mentioned in chapter 5, the original design for the syringe barrel holders needed improvement. The design for the improved holder is shown in figure 7.4. Part A is mounted on the same base block as the plastic holder was. It has a 16 mm diameter half cylinder channel into which the syringe barrel fits. The flange at the top of the barrel is a push fit into the 2 mm slot that is cut vertically into the block. This prevents any longitudinal movement. Part B is another half cylinder piece that fits over the top of the barrel, holding it firmly in place. It is screwed down with a hand tightened M4 bolt. A compression spring is positioned between the two halves to make syringe removal easier.

Summary

The design modifications that were made are summarised in table 7.1 below.

Design Modification	Problem	Solution
Syringe driver mechanism	Slider mechanism too loose	Fitting of hand tightening bolt to slider mechanism
Layout of microswitches	Both switches not being triggered by syringe driver	Addition of extension plate to syringe driving mechanism
Operational procedure	Rubber bung peeling off syringe plunger	Addition of sterile normal saline to inside of syringe barrel

Motor for 3 way tap drivers	Tap drivers stall when taps are contaminated with blood	Fitting of more powerful motor
Redesign of 3 way tap drivers	Interference from ambient infrared light	New design relocates infrared sensors inside casing
Pressure transducer amplifier	Insufficient pressure measurement range	Redesign of amplifier electronics to increase range of measurement
Syringe barrel holders	Design unacceptable for clinical use	New design that is clinically acceptable

Table 7.1 Design Modifications

At this stage of the project, the system hardware was complete. The development of the control software could therefore begin. This is the subject of chapter 8.

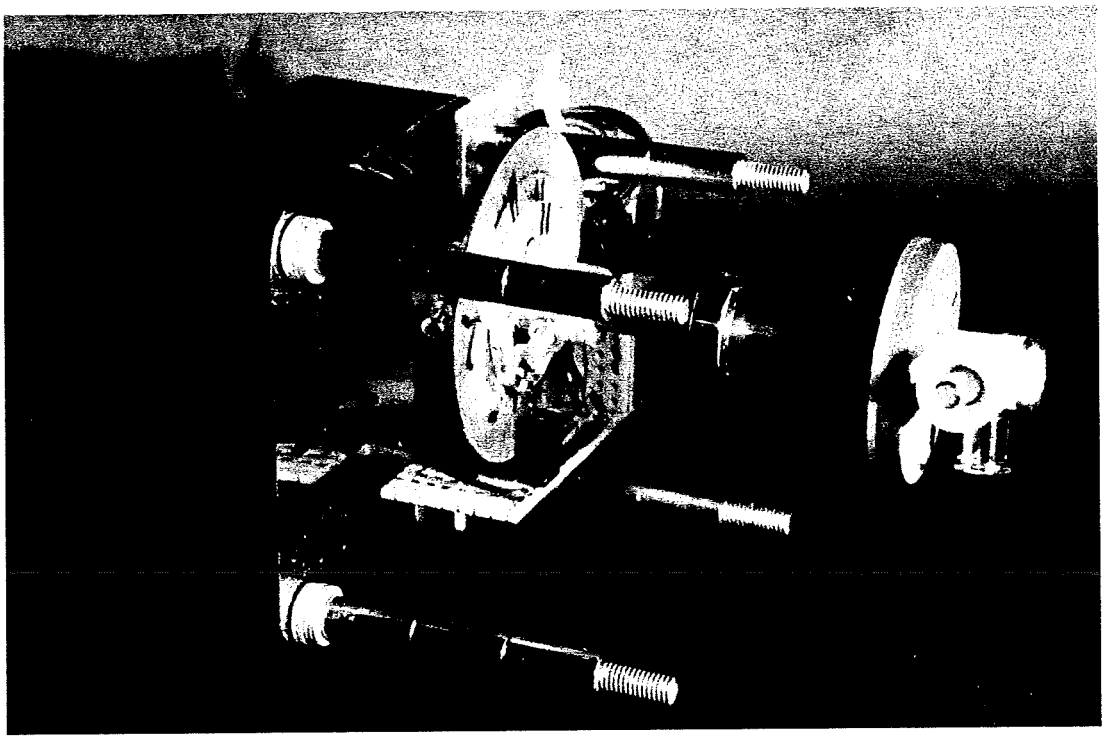


Figure 7.1 New 3 Way Tap Driver

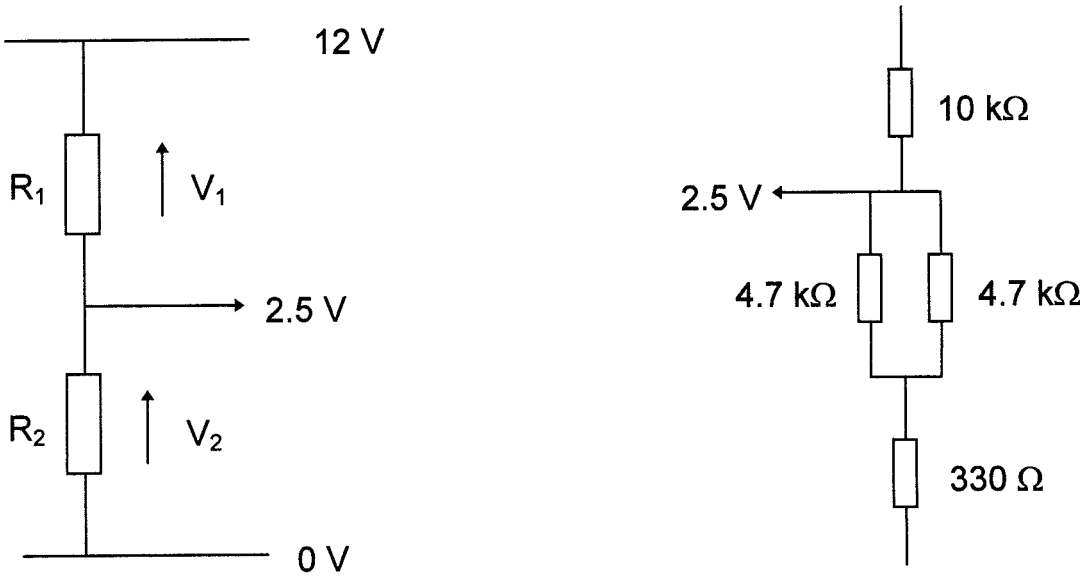


Figure 7.2 Calculation of Potential Divider for Tap Driver Circuit

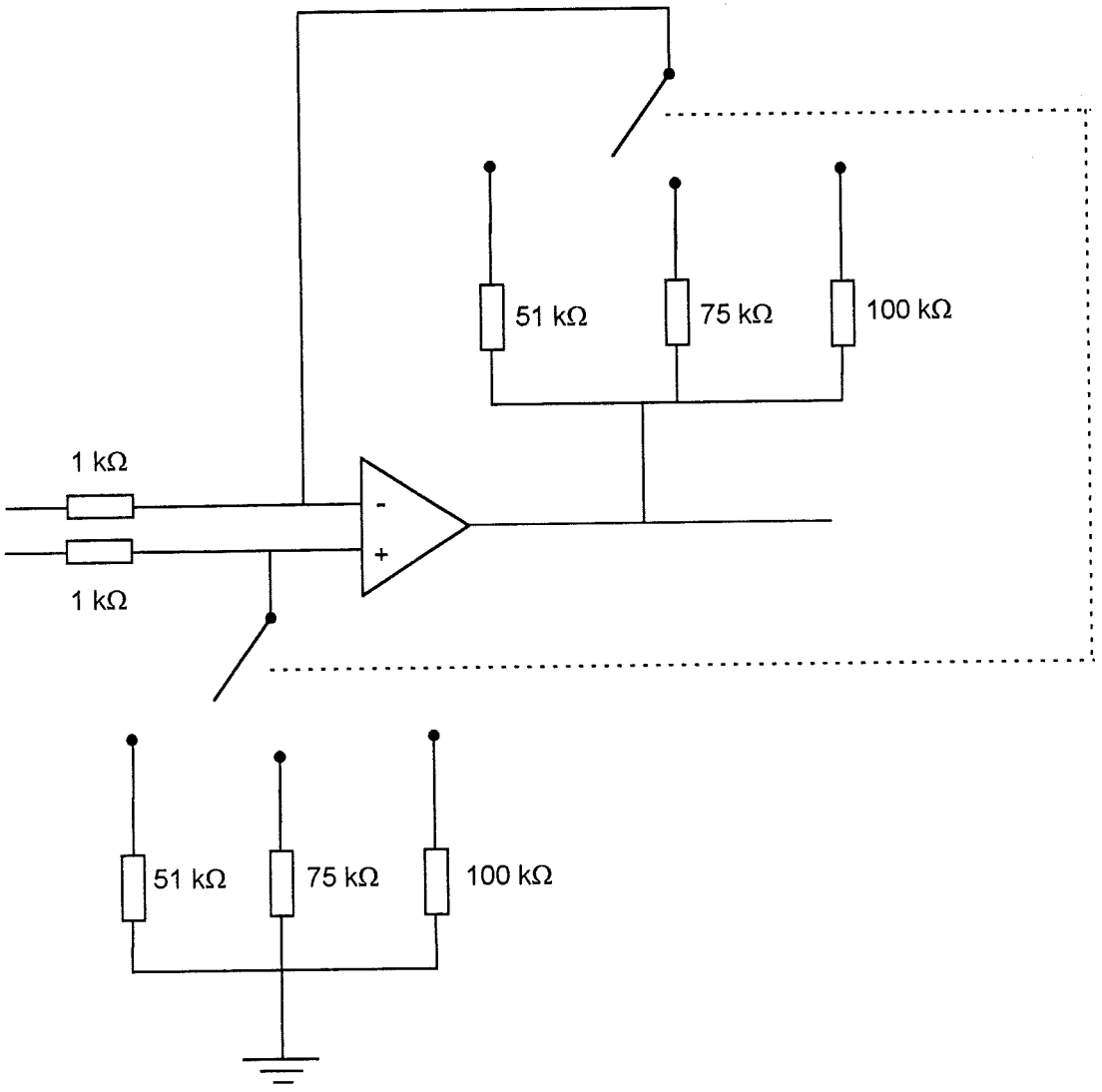


Figure 7.3 Modification to Pressure Transducer Amplifier

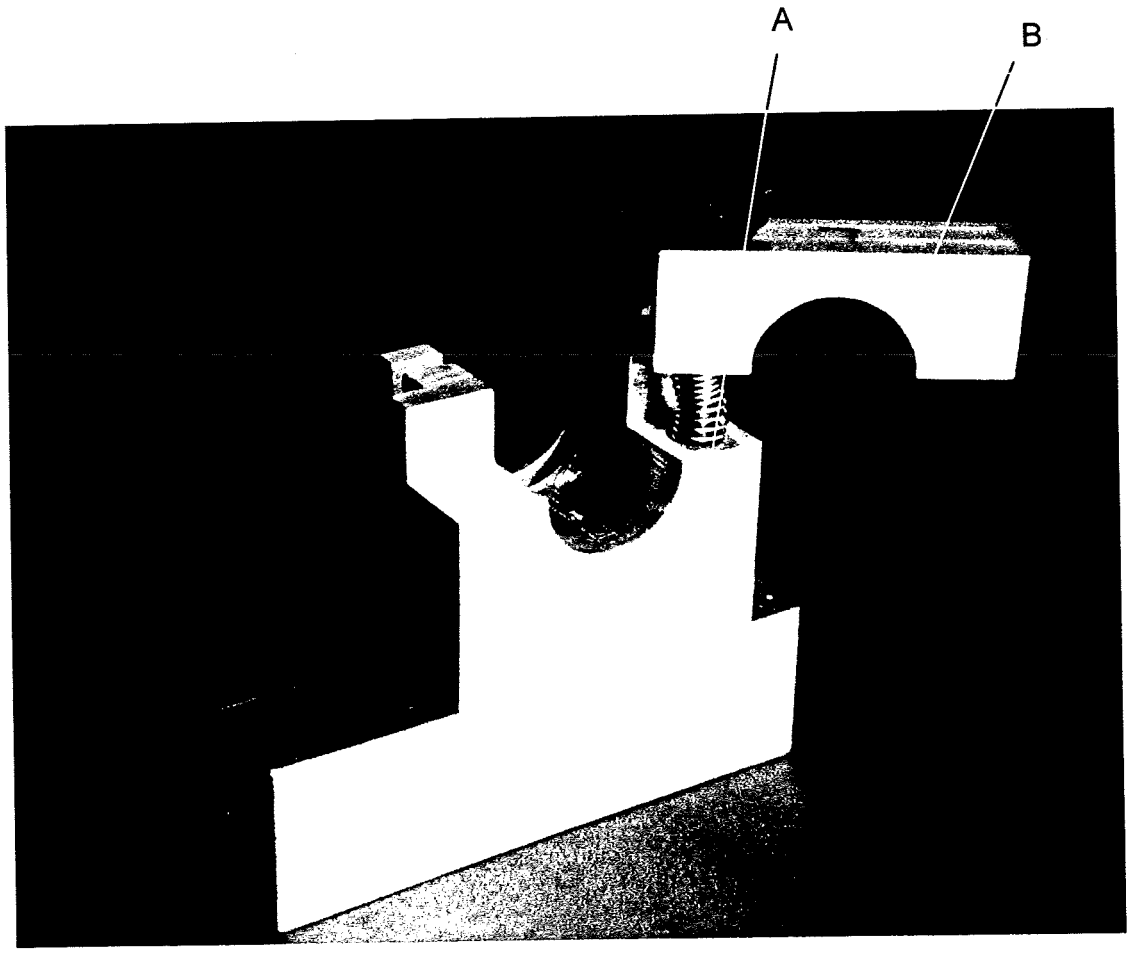


Figure 7.4 Syringe Barrel Holder

CHAPTER 8. SOFTWARE DEVELOPMENT

Introduction

The programming task was divided as far as possible into self contained units for ease of development. Each unit was developed as its own project in the LabWindows/CVI environment. The main individual projects were as follows:

Filtration (code to manage the filtration phase)
Servos (code to control the three way taps)
Withdraw and Return (control of blood withdrawal and return)
Graphical User Interface

A LabWindows project is a self contained unit with its own source code and header files, usually stored in a separate Windows directory. Where necessary, temporary interface code was written to test the control code. Once the 4 main elements had been developed and tested, they were integrated using 2 further projects. The first stage of integration was the whole control system minus the graphical user interface. The second stage added in the user interface.

The file listing of the 'whole system' project (the final stage of integration) illustrates the overall structure of the application. This is shown below.

haemo1.c
initialise.c
runsystem.c
servo2.c
withret.c
pump4.c
updtimer.c
round.c
error.c
haemo1.uir
haemo1.h
header.h

The program code is divided up into seven separate source code files. This allows each source code file to be kept to a manageable size. Haemo1.c contains the main() function and the callback functions that are attached to the control buttons on the user interface. Initialise.c contains the code that performs the system testing

and initialisation that is done when the machine is first switched on. Runsystem.c is the main algorithm that controls the operating cycle of the system. The code that controls the 3 way tap drivers is in servo2.c. Withret.c controls the blood withdrawal and return processes, and pump4.c performs the blood filtering phase of the operating cycle. Updtimer.c contains a small function that updates the timer display on the user interface. Round.c consists of a rounding function that rounds a float argument to the nearest integer. It is used by the blood withdrawal and filtering algorithms. Error processing and reporting is dealt with by the code in Error.c. The haemo1.uir file is the user interface resource file. This file contains the user interface that has been constructed with the User Interface Editor. This editor also automatically generates the file haemo1.h. This header file connects the user interface to the source code. The other file, header.h, is the header file for the source code itself.

The LabWindows Data Acquisition Library provides the connection between the source code and the machine itself, via the data acquisition card.

8.1 Data Acquisition Library Functions

Four data acquisition functions are used to communicate with the hardware. The following explains how they work.

DIG_Prt_Config(Board, Port, Handshaking, Direction)

This function takes four arguments. The first is a board number, which allows the software to access more than one I/O board. Since there is only one board in this application, this argument is always set to 1.

The 24 digital lines of the DAQ board are arranged into three 8 bit ports, A, B and C (see figure 7.10). Each port can be configured as either input, output or a mixture of both. In this application port A (or port 0) is set to output, and port B (port 1) is set to input. A direction value of 1 configures the port as output, and 0 configures it as input. Handshaking is disabled.

DIG_Out_Line(Board, Port, Line, Line State)

This function changes the state of individual digital output lines. Again the board argument is always set to 1. Since all the output lines are port 0, the port

argument is always set to 0 for this function. The arguments for Line and Line State are defined by preprocessor commands contained in the file Header.h, to make programming easier. So, for instance, DIRN_A refers to the direction line for syringe driver A (the left hand syringe), and this is line number 0. A value of 1 on this line will drive the syringe upwards, so this is defined as UP.

DIG_In_Line(Board, Port, Line, Line State)

This function reads the state of digital input lines. This time the Port argument is always set to 1, as port 1 is the input port. Again the line numbers are preprocessor defined, so SWITCH_A is defined as 4 and this is the microswitch for syringe A. The line state variable is referenced by a pointer.

AI_VRead(Board, Channel, Gain, Voltage)

This function reads the voltage input from the pressure transducer. The channel argument is always 0 as only one analogue channel is being used. The gain is set to 2 so the DAQ board can read voltages in the range 0 - 5 V. Again the input variable is referenced with a pointer.

8.2 Blood Filtration Phase - Control of Ultrafiltration Rate

Overview

Before the ultrafiltration rate control can be programmed, the basic reciprocating motion of the filtration phase has to be implemented. In the early design work it was decided that transducers which could sense the exact position of the stepper motors were unnecessary - all that was needed was a microswitch on each stepper motor that could sense when the syringe plunger had reached the bottom of the barrel.

The filtration phase begins with syringe B at the top of its travel, and syringe A at the bottom. The direction lines are set so that plunger B is travelling downwards, and plunger A is travelling upwards. Clock pulses are then issued to both stepper motors, at 0.025 second intervals, to give the required overall blood flow rate through the haemofilter of 10 ml/min. The clock pulses are issued in a

loop that also tests the state of the microswitch on the downward travelling syringe. When this microswitch changes state the direction of travel of each syringe is simply reversed, and the clock pulses continue as before. The state monitoring is transferred to the other microswitch on plunger A. When this microswitch changes state one cycle of the reciprocating motion is complete, and the cycle can begin again.

The ultrafiltration rate is the most important parameter in the machine's operation. It is the volume rate at which fluid is removed from the blood through the fibres of the haemofilter. It is a function of the blood flow rate through the haemofilter, the haematocrit of the blood and the transmembrane pressure across the fibres, i.e. the mean pressure difference across the wall of the fibre.

Two different control algorithms were written and tested, and their performance compared. Firstly a closed loop method was tried. The blood flow rate was fixed at 10 ml/min. The manufacturer's data was extrapolated to obtain an approximately linear relationship between transmembrane pressure and ultrafiltration rate. The underlying reciprocating motion of the syringes was programmed first (see above). This gave a basic rate of 40 steps/second, the syringes obviously travelling in opposite directions. A proportional control loop was superimposed on this motion. The system pressure was measured continuously, and extra steps were added to keep the transmembrane pressure constant. If the pressure was too low, extra steps were made by the pushing syringe. If the pressure was too high, they were added to the pulling syringe. Various constants of proportionality were tried to find the most effective algorithm. Good stable control was achieved.

The major drawback with this control method is that it does not take account of variations in haematocrit. So if the blood being used has a high haematocrit, the ultrafiltration rate will be too low, and vice versa. Therefore this was not a practical control system.

The ideal control system would have a transducer that could directly measure the filtrate volume output from the haemofilter. This method was considered. However, a suitable flow transducer was not available commercially, so a different design had to be used instead.

It was decided to control the blood volume in the circuit directly. Although this is an open loop method, it has a much better accuracy in practice than the closed loop method above. The method is as follows.

The blood return and withdrawal phases are timed. This is necessary because the blood withdrawal time can vary depending on how good the vascular access to the baby is. The reset time (the time taken for plunger A to travel to the bottom of its barrel and plunger B to travel to the top of its barrel at the end of the 4 minutes of filtration) is also measured and added to the total. The filtration time is fixed at 4 minutes, so an actual filtration rate can be calculated which will give the mean filtration rate required. An example of this calculation is shown in appendix B. This subject will be dealt with in more detail in section 8.5.

The reciprocating motion is the same as before. The reduction in overall circuit volume is achieved by adding extra steps in the pushing syringe at regular intervals throughout the 4 minute filtration phase. So, for example, if the filtrate volume required is 1 ml, the circuit volume is simply reduced by 1 ml during the 4 minutes of the filtration phase, by adding extra push steps. This is still not an ideal method, but it has the big advantage of almost completely compensating for variations in haematocrit. A high haematocrit will result in a higher back pressure inside the fibres, increasing the TMP and therefore the filtration rate.

Pressure monitoring is no longer part of the control method, but is still used as part of the safety system. The pressure inside the blood circuit is continuously monitored, and if it falls outside pre-set limits the machine is stopped.

Detail

The code for the filtration phase is contained in the file pump4.c. This file contains 2 functions. The function Filter_Blood() is the function that performs the filtration. Return_To_Start_Posn() resets the syringes to their starting point at the end of the filtration phase.

Filter_Blood(f_rate, volume_filtered)

This function takes 2 arguments. The first is the actual filtration rate, as explained above. The second is the actual volume filtered out during the 4 minutes. This is passed by reference back to the calling function.

This code makes extensive use of the LabWindows Utility Library timing functions. The first of these is at line 30. Timer() returns the time elapsed since the first call to any of the timing functions, so this line effectively 'starts the clock', to allow the function to be timed to run for 4 minutes. To simplify the code for plunger

movement 2 states are defined, CLOCK and ANTI. In the CLOCK state, plunger A is moving up, and plunger B is moving down, and vice versa for ANTI. Line 32 sets the starting direction as CLOCK. Lines 36 - 38 are an initial safety check. The constants FILTER_LP_LIMIT and FILTER_HP_LIMIT (defined in Header.h) set upper and lower safe limits for the pressure during the filtration phase. If these are exceeded, an error is returned and the program is halted.

Line 44 calculates the interval between filtration steps (see above). Firstly the blood flow rate is calculated in ml/h. STEP_RATE is the number of steps made per second by the stepper motors. NUM_STEPS_PER_ML is the number of steps taken to sweep out a volume of 1 ml. So the former divided by the latter gives the blood flow rate in ml/s. Multiplying by 3600 gives the flow rate in ml/h. This is then divided by the filtration rate required to give the interval (in numbers of steps) needed between filtration steps. This is then rounded to the nearest whole step.

The main algorithm consists of 2 nested *while* loops, the outer one starting at line 58. The outer loop provides the reciprocating motion of the syringes, while the inner loop provides the stepping sequence. Both these loops terminate when the function execution time has exceeded 240 seconds.

Lines 66 - 74 set up the syringe drivers to drive in a clockwise direction. Lines 80 - 88 set up for anticlockwise motion. Line 100 records the step count at the start of a syringe stroke. This is for failure testing.

The step sequencing loop starts at line 105. Line 112 provides the timing of the step sequence. The function SyncWait(beginTime,interval) is basically a delay function. It waits until the time given by the argument 'interval' has elapsed from the time given by beginTime. So in this case beginTime is the variable mark, which is the time when the outer loop starts. Since the required interval between steps is 0.025 seconds, 0.025 * step_counter will give successive marks that the motor steps can be synchronized with. Providing the code contained in the inner loop does not take more than 0.025 seconds to execute, synchronization of motor steps will be maintained. Lines 115 -117 allow the machine to be stopped by the user while this loop is being executed. This type of code will be discussed in section 8.6.

Lines 121 and 122 check the syringe driver. It takes 2500 steps to sweep a 10 ml volume of the syringe. So if this number is exceeded a fault will have occurred. This can happen in one of two ways. The first is if the stepper motor or drive board fails. Clock pulses will be issued but no movement will occur. The second way is if the microswitch on the pushing plunger fails. In this case the end of the stroke will not be detected and the step count will exceed the 2500 limit.

Lines 124 - 126 check for errors in the pulling syringe microswitch. 25 steps after the start of the stroke the state of the pulling microswitch is tested. By this stage of the stroke it should have changed state from 1 to 0. If this hasn't happened then an error is reported.

Lines 128 - 130 check the pressure inside the circuit. The current pressure is compared to the reference pressure measured at the beginning of the function. If the pressure drops or rises outside the limits defined in header.h then an error is reported.

Lines 135 - 157 are the core of the whole function. The *if* statement at line 135 tests if both syringes should be stepped or just the push syringe. Each time a clock pulse is issued the interval_counter variable is incremented. When this reaches the number stored in step_interval a push step is issued, and the interval counter is reset to zero.

The clock pulse code works as follows. The syringe_push line starts at state 1. Line139 switches it to state 0. The delay of 0.1 ms is necessary because this is the minimum pulse width that the stepper motor drive board can detect. The line is then switched back to state 1. The interval_counter variable is reset to zero. Both the step_counter and filter_step_counter variables are incremented. The latter variable is used to calculate the total volume filtered at the end of the function.

The code following the *else* statement on line 147 steps both syringes. This works in the same way as the previous code.

Finally, at the end of the inner loop, the state of both microswitches is read. The loop terminates normally when the pushing plunger reaches the bottom of its barrel. So the direction variable is changed such that the syringe directions are reversed and the outer loop begins again.

Once the outer loop has terminated, the uncorrected filtrate volume is calculated. This is done by dividing the total number of filtering steps by the number of steps that sweep out a 1 ml volume. This gives the volume filtered in millilitres. This subject will be dealt with in more detail in section 8.5.

Finally a synchronization check is done. This makes sure that the clock pulses have remained synchronized at a rate of 40 per second. At this rate, there should be 9600 pulses in the 4 minutes of the filtration phase. If the number of pulses has been substantially less than this then an error is called. This error check is important. Normally the code inside the inner loop will easily execute within the 0.025 seconds that is allowed. However, if the computer is set up incorrectly the time taken can easily exceed this limit. An example of this would be

incorrect settings in the Windows display driver. This can result in slow screen updates, leading to a loss of synchronization.

Return_To_Start_Posn()

This simple function is needed to reset the system at the end of the filtration phase. The filtration function runs for exactly 4 minutes. When it stops the syringes can be at any position in their stroke. So it is necessary to reset them to the starting position, that is with the left hand syringe (syringe A) at the bottom of its stroke and syringe B at the top, ready for the blood to be returned to the patient.

Line 206 makes the microswitch on syringe A the one that is monitored to detect the end of the stroke. Lines 209 and 211 set the desired syringe travel directions. A reference pressure is then read to use for safety monitoring inside the clock pulse loop. The same timing method is then used as in the previous function. A *while* loop provides the main body of the function. This executes as long as push_switch_state is zero, that is until syringe A reaches the bottom of its travel. Lines 226-228 allow the user to stop the system while it is executing the *while* loop. Lines 233-239 use the same code as in the previous function to step both syringes. Error checking is then done within the stepping loop. The number of steps executed is checked. If this is greater than 2500 an error is called, as this number is more than sufficient to sweep a 10 ml volume. The circuit pressure is then checked. The loop then repeats.

8.3 Blood Withdrawal and Return

Overview

The code needed for blood return is fairly straightforward, but that needed for blood withdrawal is much more complex. This is because the venous line to the baby can easily become partially or totally blocked, reducing or preventing blood withdrawal. This can happen for two main reasons - either the line becomes occluded because of blood coagulation, or the end of the line can move against the wall of the vein. It becomes blocked by attaching itself to the wall with suction. In the manual procedure, the doctor or nurse performing the blood access can feel this blockage as increased resistance in the syringe plunger. The problem can

often be overcome by reducing the rate of blood withdrawal. If this does not work, a small amount of blood can be pushed back down the line in an attempt to clear the blockage.

The algorithm for blood withdrawal had to reflect the procedure used by the clinician as closely as possible. An attempt was made in consultation with the RVI to quantify this procedure as far as possible. As a starting point for the programming, the following parameters were decided on :

if 0.5 ml of plunger movement results in no blood being withdrawn, then the plunger should stop to allow time for the blockage to clear. If the blockage does not clear, then eventually the blood flow should be reversed, but no more than 0.5 ml of blood should be pushed back down the venous line.

The pressure drop obtained with 0.5 ml of plunger movement (with the venous line clamped obviously) was measured. This was found to be equivalent to a 1.3 V change in the output from the pressure transducer amplifier. This represents a pressure drop of 520 mm Hg The initial withdrawal algorithm was based on this pressure drop limit.

The number of steps needed to withdraw the syringe 10 ml was determined, so that the volume swept by each step could be calculated. This was done by first measuring the length between the 0 ml and 10 ml marks on the syringe being used. This was found to be 61.5 mm. So a 1 ml volume is swept by a movement of 6.15 mm. Each step of the stepper motor produces a linear movement of 0.025 mm. Therefore the number of steps to sweep 1 ml is:

$$\frac{6.15}{0.025} = 246 \text{ steps}$$

So each step is equivalent to 1/246 ml. The blood volume to be withdrawn from the patient is entered into the graphical user interface (this is known as the Working Blood Volume). This figure is then divided by 1/246 to obtain a figure for the total number of motor steps required for the given volume of blood to be withdrawn.

At the start of the withdrawal procedure, the pressure inside the circuit is measured, to obtain a reference pressure. The plunger is then moved upward, at a rate equivalent to a blood withdrawal of 10ml/min. If the pressure in the syringe drops by more than 520 mm Hg, the stepper motor is stopped, initially for 1 second. If the pressure rises by more than 20 mm Hg in this time, then blood

withdrawal starts again. If the pressure does not rise, then there is a further delay of 1 second. If 4 seconds go by without a rise in pressure, a more serious blockage is implied. In this case the flow in the venous line is reversed, and a volume of 0.5 ml is pushed back down the line. Blood withdrawal then begins again.

Initially 2 different modes of blood withdrawal were programmed. The above procedure is one. The other is a variation on it. The delay and reverse flow code is the same. Here though, every time 2 reverse flows have been triggered, the blood withdrawal rate is halved. So after 4 reverse flow function calls, the blood withdrawal rate will be reduced to 2.5 ml/min.

There is an overall time limit on blood withdrawal that is proportional to the blood volume being withdrawn. If this time limit is exceeded, it is assumed that the venous line is irreversibly blocked. The machine will be stopped and the alarm set off.

Detail

The code for blood withdrawal and return is contained in the file withret.c. As well as the 2 main functions, there is also a function to reverse the flow of blood during the withdrawal procedure. These 3 functions will be dealt with in turn.

Withdraw_Blood(blood_volume)

This function takes just one argument, the volume of blood to be withdrawn - this is picked up from the user interface. Line 30 gets the access mode from the user interface. This is just an integer which indicates which of the alternative access algorithms is to be used. A reference pressure is then read. Lines 38 - 40 check that syringe B is at the bottom of its travel. If the microswitch is at state 0 an error is reported. Syringe B is then set to travel upwards. The number of steps needed to withdraw the required amount of blood is then calculated. A time limit is set for blood withdrawal. 30 seconds is allowed for each ml of blood to be withdrawn. A starting time for the function is then put into the variable mark. The main loop for the function begins at line 64. The loop tests that the required volume hasn't been reached yet and that the withdrawal process hasn't exceeded the time limit. As before, lines 69 - 71 allow the user to stop the system while this loop is being executed. Lines 77 to 80 check that the syringe microswitch is working

properly. The pressure in the circuit is then checked against the overall pressure limits allowed, stored in WITHRET_P_DROP and WITHRET_P_RISE.

The *if* statement at line 94 deserves some explanation. It pairs with the *else* statement at 124, so the code in between is that executed if the conditions aren't right to continue blood withdrawal. The *if* statement is complicated because of the physical compliance inherent in the system. When the pressure exceeds the 520 mm Hg limit the plunger is stopped for 2 seconds. Even if the venous line is totally blocked, there will tend to be a slight pressure rise during the first 2 second delay. So when the pressure is tested again, it must have increased by more than 20 mm Hg from the previous value for blood withdrawal to begin again. This is achieved by the two statements on either side of the OR operator. The delay_counter variable allows the *if* statement to differentiate between the two situations.

So if a blockage is detected, the code from line 98 onwards is executed. Initially the code following the *else* statement on line 106 will be executed. A wait of 1 second is introduced. The delay_counter variable is then incremented. If a wait of 4 seconds has not allowed the blockage to clear, then the Reverse_Flow() function is called. The number of times this function has been called is recorded in the reverse_flow_count variable, and the delay_counter variable is reset to zero.

Access mode 1 does not change the withdrawal rate. Access mode 2 does. This is achieved by the code segment from line 98. If reverse flow has been triggered more than once, then the speed_factor variable (which starts off as 1) is multiplied by 2. This halves the rate at which blood is withdrawn.

Once any blockage has been cleared blood withdrawal can continue normally. This is achieved by the *else* statement at line 124. This is the standard code to issue a clock pulse to the stepper motor, apart from the addition of the speed_factor. The delay times are multiplied by the speed factor to lengthen the clock pulses and slow the motor down. The *if* statement at line 142 returns an error if blood withdrawal has exceeded the time limit set for it.

Return_Blood()

This is a simple function that returns the blood to the baby after the Return_To_Start_Posn() function has executed. Line 168 reads the syringe B microswitch to make sure the *while* loop doesn't execute if the plunger is already at the bottom of the barrel. The direction of plunger travel is set to downwards. A

reference pressure is then read for safety monitoring. The subsequent code is the same as that used in previous functions to perform syringe stepping. Only syringe B is being moved. The total number of clock pulses issued is checked at line 194 to make sure it hasn't exceeded the usual 2500 limit. The pressure inside the circuit is also checked every time the loop executes.

Reverse_Flow(step_counter, v_ref)

This function is called from inside Withdraw_Blood(). It pumps 0.5 ml of blood back up the venous line when a blockage is detected. It takes two arguments. The first is the variable step_counter, passed as a pointer. The Withdraw_Blood() function needs to keep track of the exact number of steps that have been made during the withdrawal procedure. This is done using the step_counter variable. It is decremented every time a reverse step is made, and the final value is passed back to the calling function. The second argument is v_ref. This is the reference pressure that was read at the start of the Withdraw_Blood() function. This is needed because the Reverse_Flow() function will not be called at ambient pressure so it would not be possible to obtain a reference pressure while inside the function.

Line 228 sets syringe B to move downwards. The B microswitch is read as usual before the main loop starts. There are 2 conditions for the *while* loop. The first one tests the number of steps that have been made. Since a 1 ml volume is swept by 246 steps (see earlier) 0.5 ml corresponds to 123 steps. The microswitch state is also tested, so that the loop does not try to move the plunger beyond the lowest point of its travel. The usual code for stepping follows, with 2 additions. The step_counter variable is decremented as explained above, and the reverse_counter, the variable that is used in the *while* loop test, is incremented. When loop execution has finished, the syringe direction is set back to UP, so that the system leaves the function in the same state that it entered it.

8.4 Three Way Tap Driver Control

Overview

As was mentioned in chapter 5, the original design called for 3 target positions at 90° intervals. The problems with intermediate codes caused by edge detection made the programming very complicated. Although a workable program was produced, it was not reliable enough. So the specification was changed to just 2 target positions 90° apart.

The connection between the hardware and the software is provided by 4 control lines for each tap driver, 2 input and 2 output. The 2 input lines are connected to the 2 photodetectors. They are at state 0 when the slot in the aluminium disc is underneath the detector, and at state 1 otherwise. The 2 output lines give clockwise and anticlockwise movement of the driving disc. They are active high.

The first step in the control algorithm is to read the photodetectors to determine the current state of the tap driver. This can be one of three states. The disc slot can be under either one of the detectors (designated states 1 and 2), or under neither of them (state 3). The direction of rotation needed for the shortest route to the desired position is then calculated. The motor is driven in the desired direction by a series of short pulses (0.02 seconds long). After each pulse the photodetectors are read to see if the driver has reached the target position. When the position is reached the pulses are stopped. Because of the high gear ratio (500:1) and the frictional resistance in the 3 way taps, there is very little overshoot of the target position, and this simple method provides sufficient positional control.

Detail

The code for tap driver control is contained in the file servo2.c. Because of the repetitive nature of the algorithm, it was divided up into 4 separate functions to make the code more efficient.

MoveServo(Servo_Num, Target_Posn)

This function takes two integer arguments. The first identifies which tap driver is to be moved, 1 or 2. The second argument gives the position to be moved to, also 1 or 2.

The *switch* statement from line 36 uses the servo number as its *case* variable. The *case* statement starts by assigning the input line numbers for the servo to be moved to the variables i_line1 and i_line2. The line numbers are defined in the header file header.h. The difference between the current position and the target position is then calculated, to find out which direction of movement will supply the shortest path to the required position. The output line to be driven is then assigned to the variable output_line. If the target position and the current position are the same the anticlockwise drive line is assigned. This is arbitrary, as either line could be chosen. An output line is assigned so that a no movement position check can be performed (see later). The current position is worked out by a separate function that will be dealt with later. The function returns the current position of the servo, that is 1 or 2.

A variable state_11 is used as an error checking device. It checks that the detector input lines haven't failed to a zero state. The variable itself is a logical one - it is given the value 1 if both input lines reach the value of 1 during servo movement. Before movement begins it is assigned the value zero (line 80). Another logical variable, no_movement, is also used for error checking. If the current position is the same as the target position then it is set to TRUE to indicate that no movement will take place.

The main code consists of two nested *while* loops. The inner loop (from line 87) executes while the servo is looking for the target position. It starts with a 1 being written to the drive line, starting the servo motor. There is then a wait of 0.019 seconds, and then the motor drive is switched off. The *if* statement from line 94 tests the detector input lines. The Current_Posn() function returns a value of 3 when both detector lines are at state 1. If this happens then the state_11 variable is assigned the value of 1. The outer *if* statement prevents this code being executed needlessly every time through the loop. The pulse_counter variable is then incremented. If it exceeds 1000, execution of the loop is terminated, as it is assumed that a fault has occurred.

The outer *while* loop allows for a change of direction of rotation while the target position is being sought. This is useful in the unlikely event that the servo

overshoots the target position. The pulse counter is reset to zero, and the function Change_Direction() is called. This function changes the active output line from clockwise to anticlockwise and vice versa. A cycle_counter variable monitors how many changes of direction occur. If this exceeds 8 the loop is terminated.

Lines 109 -112 report errors. If the servo has searched back and forth for the required position more than 8 times an error is reported. If there has been servo movement but no intermediate state where both input lines were at logic 1, an error is also reported.

The code from line 117 deals with the situation where the target position is the same as the current position. It is necessary in this case to check that the servo really is at the position that the detectors report, because detector failure would give a false position indication. This is done by the function Check_Servo_Posn() (see below).

Current_Posn(i_line1, i_line2)

This function takes the 2 current detector input lines as its arguments. The state of the lines is read into the variables state1 and state2. A series of *if* statements then assign codes to the variable Current_Pos, corresponding to the three possible states of the input lines. This code is then returned to the calling function.

Change_Direction(*o_line)

This function takes the active output line as its argument. It is passed by reference so that the function can change the value and pass the new value back to the calling function. The value is changed by a *switch* statement. If the value passed in is a clockwise line, it is changed to an anticlockwise line and viceversa.

Check_Servo_Posn(i_line1, i_line2, output_line)

This function takes three arguments. The first two are the currently active detector input lines, and the third is the currently active output line. It works by moving the tap driver a small distance away from its current position. The detector

inputs are then checked to make sure that they are both at state 1. If this condition is satisfied the tap driver is then moved back to its previous position.

Line 174 puts the initial position into the variable start_posn. The first *while* loop moves the servo until position 3 is reached, i.e. both detectors at state 1. Again a pulse counter variable is used here. However the error limit is much lower as the state change should occur with very little movement.
After the first loop has executed, the direction of rotation is changed and the second *while* loop executes. This drives the servo until it returns to its original starting position.

8.5 Software Integration

Overview

The functions described up to now control the different elements of the system's operating cycle. The code that integrates these different elements is contained in the function run_system().

The elemental functions are called in sequence to produce the basic cycle of blood withdrawal, blood filtration and blood return. The other major task of the integrating function is to calculate the actual filtration rate that needs to be passed to the Filter_Blood() function to give the mean filtration rate entered at the user interface. Various user interface screen updates are also performed.

Detail

The code for the run_system() function is contained in the file RunSystem.c.

run_system(blood_volume, mean_f_rate)

This function takes 2 arguments. The first one passes in the working blood volume, that is the volume of blood that will be withdrawn from the patient during the withdrawal phase. The second passes in the mean filtration rate, that is the filtration rate that is entered at the user interface.

The function starts with the global_stop variable being set to zero. This is necessary in the circumstance where the system has been stopped and restarted.

This variable will be explained fully in section 8.6. The system timer on the user interface is reset to zero by the Reset_Timer() function on line 24. This function will be covered in section 8.9.

The following explanation of the control code can be followed with reference to figure 4.2.

The first two calls to MoveServo() move the 3 way taps into the positions needed to allow the syringes to be reset. This is tap 2 open and tap 4 closed. All the main function calls in this file are followed by a check on the return code to make sure an error has not occurred inside the called function.

The main control loop starts at line 40. The *while* condition is TRUE because this loop only terminates when the stop button on the interface is pressed or an error occurs. The time at the start of the loop is put into the variable off_time_start. The 'off time' is the time spent in the loop not filtering blood. Lines 48 and 49 terminate the loop if the stop button was pressed during the previous execution of the Filter_Blood() function. The Update_Timer() function refreshes the Treatment Time display on the user interface. This function will also be covered in section 8.9. The cumulative filtrate volume display is then updated. This display gives the total volume of filtrate that has been separated from the blood in the current treatment session. Its value is stored in the variable cum_f_volume. This variable is incremented by line 57. The volume filtered by the last execution of the Filter_Blood() function is added to the running total, after being divided by the filter correction factor. This factor needs some explanation. The volume of filtrate passing out of the haemofilter is in fact less than the reduction in volume of the circuit during the filtration phase. The reasons for this will be explored in chapter 10. So to achieve the required filtration rate, the filtration rate entered into the user interface is multiplied by a correction factor, worked out by experiment to be 1.104. So, to achieve a reasonably accurate figure for the filtrate volume as displayed on the user interface, the volume_filtered variable passed back from Filter_Blood() must be divided by this same factor.

The SetCtrlVal() function performs the actual screen update. The function ProcessSystemEvents() on line 60 is a User Interface Library function that makes sure that screen updates keep time with program execution.

The Return_To_Start_Posn() function is then called. This moves syringe A down and syringe B up ready for blood return. A 5 second delay is then called. This allows any pressure gradient that may have built up during filtration (on the previous pass through the *while* loop) to equalize. The tap drivers then move into

position to allow blood return. This is tap 2 closed and tap 4 open. The Return_Blood() function is then called. Line 102 gives a 3 second delay, again to allow for pressure equalization. Blood is then withdrawn from the patient, followed by another delay, this time of 4 seconds.
The taps are moved into the filtering positions - tap 2 open and tap 4 closed.

The actual filtration rate needed is then calculated, as was discussed in section 8.2. Line 129 calculates the total time spent in the *while* loop up to this point, by subtracting the variable off_time_start from the current time. So the value of off_time is the total time spent not filtering blood in one operating cycle. Since the time spent filtering blood is always 240 seconds, the actual filtration rate needed can be calculated by multiplying the mean filtration rate by $\frac{240 + off\_time}{240}$. This is done in line 132. This calculated value is then passed to the Filter_Blood() function on line 140.

The Filter_Blood() call is the last line of the main *while* loop. The remaining lines, 154 to 159, perform final screen updates when the system is stopped.

8.6 User Interface and Callback Functions

Overview

The graphical user interface was constructed using the Labwindows/CVI user interface editor. The finished article is shown in figure 8.1. Objects are chosen from the interface library and placed on the main panel as required. They are linked to the C code via callback functions. Each object can have a callback function name assigned to it. When the object is activated (e.g. a button is pressed) that function is called and the code it contains is executed. A large library of user interface functions is available to perform various operations on the user interface. The interface itself is contained in the user interface resource file, haemo1.uir. The interface editor automatically generates a header file, haemo1.h, that links the interface to the rest of the source code. The callback functions are contained in the file haemo1.c.

The main control buttons are at the top left of the interface. Initialise System performs the setting up and testing of the system when it is first switched on. Start/Restart Treatment is self explanatory. The Change Filtration Rate button allows the filtration rate to be changed without having to reinitialise the machine.

Below this are the Stop Treatment and Quit buttons. Below these are some display boxes which give the elapsed treatment time at current settings, and the total volume of ultrafiltrate that has been produced at the current settings. The large white panel at the top of the screen displays messages from the system. At the bottom of the screen are three numeric panels into which the main parameters for the treatment session are entered. The weight of the baby to be treated is entered into the leftmost box. A standard clinical calculation is then done to arrive at the working blood volume to be used. The user then has the choice to use this figure or enter their own figure. Finally the ultrafiltration rate required is entered into the right-hand box.

The panel at top right gives treatment history. Every time different treatment settings are entered a new entry will appear in this box, giving start and stop time, ultrafiltration rate and total filtrate volume.

Pop Up dialogue boxes are also employed to interact with the user. There is a red alarm Pop Up box which appears when an alarm goes off - an audible alarm coincides with the appearance of this box.

Modifications made to User Interface

As clinical testing proceeded, various improvements were made to the user interface (refer to figure 8.1). A display of the system pressure was added (centre of screen). This is in the form of a continuous strip chart. This has proved extremely useful in practice, as it displays the pressure in real time, allowing a close eye to be kept on system operation. Various pressure controls were added, as it became apparent that it would be necessary to change the values of these controls while the system was in use. They are on the right hand side of the screen. The withdrawal trigger pressure is the venous line pressure at which the blood withdrawal rate is reduced. Below this are the overall pressure limits that are in operation during the withdraw and return phases of the operating cycle. If these limits are exceeded an alarm will go off and the system will stop. Finally, there are the pressure limits for the filtration phase. At the bottom right of the screen is the withdrawal rate display. This is an indicator only, values are not entered into this box. It displays the current blood withdrawal rate.

Detail

main()

This is contained in the file haemo1.c. Lines 15 to 20 before main() include the header files for all the Labwindows libraries that need to be accessed. Main() starts with the DAQ board ports being configured for output and input as necessary. Software_init and hardware_init are two flags that are used in the system_init() function. The main parent panel and two child panels, RESPONSE and ALARM are then loaded into memory. The RESPONSE panel is a popup panel that is used to get yes or no answers from the user. The ALARM panel is also a popup panel that appears when an error occurs.

Each LoadPanel() call returns a handle so that each panel can be identified in subsequent UI library function calls. An initial message is then displayed using the series of InsertTextBoxLine() calls. This function displays a line of text onto the selected control, in this case the message display panel.
Line 58 displays the main panel on the screen. Lines 59 to 64 then initialise the panel by dimming controls that should not be operable at this point. The SetCtrlAttribute() function is a typical UI function that takes four arguments. The first is the panel handle, which identifies the panel to be operated on. The second identifies which control on that panel is to be operated. The third is the attribute of the control to be changed, and the fourth is the new value of the attribute. In this case 1 will grey out the button, and 0 will make it active. Lines 69 to 74 get the system date and time and convert it into a suitable format to be displayed on the first line of the treatment history box. The function SetCtrlVal() displays the string on the user interface. The final line of main() runs the user interface and sends interface events to the relevant callback functions.

Callback Functions

Every callback function has a standard format and argument list. The arguments allow information to be passed from the user interface into the function. When a button is pressed, several events are generated, not just one. These correspond to such things as the button being pressed down, as well as the button being released. To prevent the callback code being executed more than once, all

the code is contained within an *if* statement, the condition of which is that the event must be an EVENT_COMMIT. This corresponds to the control button being released after a single mouse click.

System_Init()

Lines 99 to 103 dim all the control buttons so they cannot be operated while this function is executing. The *if* statement from line 108 allows some flexibility in the initialisation process. If the system hardware has already been initialised the user can choose just to reenter the treatment parameters. The library function GenericMessagePopup() is used to ask the question. Lines 119 - 120 call the init_hardware() function if the hardware_init flag is FALSE. This function will be dealt with in section 8.7. If the return code from this function is non zero then an error is called. A return code of zero indicates successful initialisation and the hardware_init flag is set to TRUE.

The init_software() function is called inside a *do-while* loop. This allows it to be called repeatedly if the user is not happy with the entered parameters. The library function ConfirmPopup() displays the text message together with yes and no buttons. A value of 1 is returned if yes is pressed, and 0 if no is pressed. Once the software has been successfully initialised the software_init flag is set.

When initialisation is complete, a final text message is displayed. All the control buttons apart from Stop are undimmed ready for use. Finally the Start button is made the active control ready for treatment to begin.

System_Start()

When the start button is pressed, the Change Rate and Stop buttons are undimmed as they must be active. Quit, Start and Initialise are all dimmed so that they are inactive. SetCtrlVal() is used to toggle the indicator controls on the left hand side of the screen to indicate that the system is running. This is confirmed by a text message on the main display.

The software parameters are obtained from the user interface and loaded into the variables blood_volume and filter_rate. The current time is loaded into start_time to be used later on the treatment history display. The filtration rate entered into the user interface is then multiplied by the filter correction factor

discussed earlier. The function run_system() is called. If this returns with a code greater than zero an error is called. Lines 214 and 215 record the finishing time and load this and the start time into a string. This is then displayed in the treatment history box. Lines 216 to 221 get the session data from the interface, format it into a string and then display it.

Change_Filter_Rate()

This function allows the filtration rate to be changed without having to reinitialise the system.

Firstly the indicator panel is toggled by lines 242 and 243 to indicate that the system has been stopped. The current filtration rate is then obtained from the user interface by GetCtrlVal(). The variable global_stop is set to 1. This is necessary to make sure that the Run_System() function terminates when the Change Filtration Rate button is pressed. Lines 254 to 258 dim all control buttons so they cannot be operated while this function is executing. Text lines are then displayed to prompt the user to enter the new ultrafiltration rate. Line 270 changes the filter rate display box from indicator mode to validate mode. This allows a new value for the rate to be entered into the box. The actual entering of the new rate is done inside a *do-while* loop, to make sure a new value is entered. Line 273 makes filter rate the active control. GetUserEvent() waits until a value has been entered and the return key has been pressed. GetCtrlVal() loads this new value into new_filter_rate. The filter rate control is then set back to indicator mode so it cannot be altered. Line 285 clears the message box. The new filtration rate is then displayed and the user is asked to confirm that they want to use the new rate. If the change is confirmed then the display reflects this. If the change is not confirmed the user is informed that the rate remains at the old value. The user is then prompted to restart the system. The control buttons (apart from Stop) are undimmed, and the Start button is made the active control.

System_stop()

This function stops the system. The global_stop variable is set to 1. Again this is important to make sure that whatever function is being executed at the time the Stop button is pressed, that function terminates properly. The message box is cleared, and a message confirming the stop is displayed. The left hand indicator

panel is toggled as usual. The Stop button is dimmed. All other control buttons are undimmed.

Quit_system()

This function terminates the user interface. A call to QuitUserInterface() causes the RunUserInterface() function to return, and the application terminates, returning control to the computer's operating system.

8.7 System Initialisation

Overview

When the system is first switched on a thorough test must take place to make sure everything is working correctly. The stepper motors and tap drivers need to be moved to the correct positions to allow the blood circuit to be attached to the machine. Everything is then retested with the blood circuit in place. The software initialisation involves the user entering the treatment parameters for the session. These are the working blood volume and the ultrafiltration rate.

Detail

The code for initialisation is contained in the file initialise.c. The two main functions that are called from system_init() are init_hardware() and init_software(). Init_hardware() in turn calls three more functions: test_hardware(), test_stepper_motor() and test_transducer_and_line().

Init_hardware()

This function starts by clearing the message box and prompting the user to remove the blood circuit from the machine if it is still attached. There are many calls to ProcessSystemEvents() in this function. This UI library function makes sure that screen updates are synchronised with code execution. If these calls are not included screen updates tend to lag behind. A popup box then asks the user to confirm that the circuit has been removed. A prompt is then issued to switch the

machine on, followed by the usual confirmation popup box. A message then confirms that the system check has begun. The first call to test_hardware() is then made (more details below). This function has one argument. The value is either CIRCUIT_IN or, in the first call, CIRCUIT_OUT. This allows two slightly different testing routines to be performed. If the first call to this function returns with no errors, a message confirms that the system is working properly. The user is then prompted to attach the blood circuit to the machine, again followed by a popup confirmation box. The test_hardware() function is called again, this time with the CIRCUIT_IN argument. If this function returns normally the test_transducer_and_line() function is called. This function is covered later on in this section.

test_hardware(circuit_status)

The two tap drivers are tested in turn by moving them back and forth between their two target positions. If the blood circuit is not attached, the two stepper motors are tested. Both 3 way taps are set to the open position before this is done. This is a precaution just in case the blood circuit has been left attached to the machine. The taps being open prevents a dangerous pressure build up that would happen if the circuit were still in place. The stepper testing is done by two calls to test_stepper_motor() (detailed below). After this the 3 way taps are moved back to their previous positions. The output voltage from the pressure transducer amplifier is then read. A series of *if* statements check this voltage against limits that are stored in header.h. If the voltage with the transducer unplugged is greater than 0.8 V an error is returned. With the transducer plugged in, the acceptable range of output voltage is 2.8 V to 3.2 V.

test_transducer_and_line()

The taps are moved into position to allow blood withdrawal from the patient, that is tap 2 closed and tap 4 open. The Withdraw_Blood() function is then called with an argument of 1.0, so that just 1 ml of blood is withdrawn. This tests that the venous line is clear. Tap 4 is then closed, and a reference pressure reading taken. The syringe B plunger is moved down 20 steps, using the standard code for this. Another pressure reading is taken. The pressure change is compared with the ONEML_P_CHNGE constant. If the change is less than this, an error is returned.

The plunger is then moved up 40 steps, to produce a pressure drop equivalent to the pressure rise previously obtained. The pressure drop is again compared to ONEML_P_CHNGE. In this way the functioning of the pressure transducer is thoroughly tested. Finally the plunger is moved down 20 steps back to its starting position. Tap 4 is opened and the blood is returned to the patient. Both taps are then returned to the positions they were in at the start of the function.

test_stepper_motor(motor_num)

This function is passed one argument, the number of the stepper motor to be tested. The initial *if-else* statement assigns the appropriate digital line numbers to the 3 variables switch_line, clock_line and dirn_line. The current Timer() value is then loaded into the mark variable in the usual way, and the direction line is set to move the stepper motor downwards. The stepper motor microswitch state is read, and then the *while* loop from line 232 moves the stepper motor to the bottom of its travel. Line 246 checks that the step count hasn't exceeded the limit of 2500. This checks both the stepper motor and the microswitch. The direction of travel is then reversed and the motor moves upwards 25 steps. This should be sufficient to open the contacts of the microswitch, and lines 268 to 270 check that this has happened. The direction of travel is then reversed again and the motor is moved back to the bottom limit of its travel. Again the number of steps that have been executed is checked.

init_software()

This function allows the user to enter the treatment parameters for the session.

The message box is cleared and the user is prompted to enter the weight of the baby to be treated. The birth weight control is then set to validate mode so that a number can be entered. The number is entered under the control of the *do-while* loop from line 449 to 455. Line 458 sets the control back to indicator mode so it can't be altered again. A standard clinical calculation for the safe blood volume to be withdrawn is then done and the result is assigned to the variable blood_volume. The calculation is done as follows: it is assumed that a typical baby has a blood volume of 85 ml per kg of body weight. So this figure is multiplied by the entered birth weight (after dividing by 1000) to give an estimate of the total blood volume. It

is assumed that it is safe to withdraw 6 % of the total blood volume. So the previous result is multiplied by 0.06 to arrive at a final value. This value is then displayed on the working blood volume control. The user is prompted to decide whether they want to use the calculated figure or enter their own figure. Lines 472 to 491 deal with these two options in the usual way. Line 494 sets the working blood volume control back to indicator mode.

Next the user is prompted to enter the ultrafiltration rate required. This code is very similar to that used above and needs no explanation.

8.8 Error Processing

Overview

Error checks are distributed throughout the control program. All possible failure states of each piece of circuitry and hardware have been accounted for in the error programming (this is discussed in more detail in chapter 9). If an error is encountered, an error processing function is called. The machine is stopped, the pop-up alarm panel appears and a continuous audible alarm is sounded. A short description of the error appears in the message box on the user interface.

Detail

All the error processing is dealt with by a single function, process_error(). This is contained in the file error.c.

process_error(error_code)

This function has one argument, the number code of the error to be reported.

The function starts by toggling the indicator panel on the left of the screen to indicate that the system is stopped. Appropriate control buttons are then dimmed or undimmed depending on whether treatment parameters have been entered or not. The message box is then cleared.

The main body of the function consists of a *switch* statement which uses the error code passed in as the control expression. There are 21 error codes

altogether. The code for each case is the same. A line of text is inserted into the message box reporting the error code number and giving a short description of the fault. This is followed by a *break* command to ensure only one *case* statement is executed. Line 200 then displays another line of text confirming that the system has been stopped. The red alarm pop up panel is then displayed. This panel contains a button which can be pressed to turn off the alarm. This works via the *do - while* loop from line 210. The loop contains a GetUserEvent() function which checks to see if the reset button has been pressed. The Beep() function issues a short pulse to the PC speaker. This loop will continue to execute until the reset button has been pressed, so the alarm sound will be continuous.

After the alarm is reset the alarm pop up panel is removed, after a short delay.

8.9 Miscellaneous Functions

Update_Timer(start_time)

This function is contained in the file UpdTimer.c. It updates the user interface at the end of each filtration cycle, giving the elapsed time since the start of the current treatment session. The time at the start of the current treatment session is passed in as the argument start_time. Line 25 then calculates the elapsed time in seconds with a call to the Timer() function to find out the current time. The value in seconds is then converted into minutes and hours. Calls to SetCtrlVal() then display the calculated values on the user interface.

Reset_Timer()

This function is also in UpdTimer.c. It resets the elapsed time display every time the system is restarted. This is done with 2 calls to the SetCtrlVal() function.

Round(number)

The source file for this function is Round.c. It is called from Filter_Blood(), and is used to round the variable float_step_interval up or down to the nearest

integer. It uses the maths library functions floor() and ceil(). The function is a general one that will return correct values for either positive or negative numbers.

8.10 Header Files

Haemo1.h

This header file is generated by the Labwindows user interface editor and cannot be altered by the user. It has 2 main sections. The first is a series of define statements that define constants for each of the user interface panels and controls. The second section contains the function prototypes for the callback functions.

Header.h

This starts with include commands for all the library header files that are needed. Lines 13 to 22 define constants that are used in the filtration functions. The CIRCUIT_IN and CIRCUIT_OUT constants are used in the initialisation functions. Lines 33 to 44 give constants for the servo functions, defining the output and input digital line numbers. Lines 49 to 70 are the function prototypes. A few global variables are then defined. The machine parameters are a series of constants which define the operation of the machine itself. Gathering them together here makes programming changes much easier when any machine redesign or improvement is made.

Summary

This chapter has dealt with the development of the software used to control the system. In summary, figure 8.2 graphically illustrates how the main functions relate to each other, so that the overall structure of the program can be seen. The completion of the software development meant that the system was now ready to be tested as a whole for the first time. The initial testing that took place is detailed later, after a review of system safety, which is the subject of the next chapter.

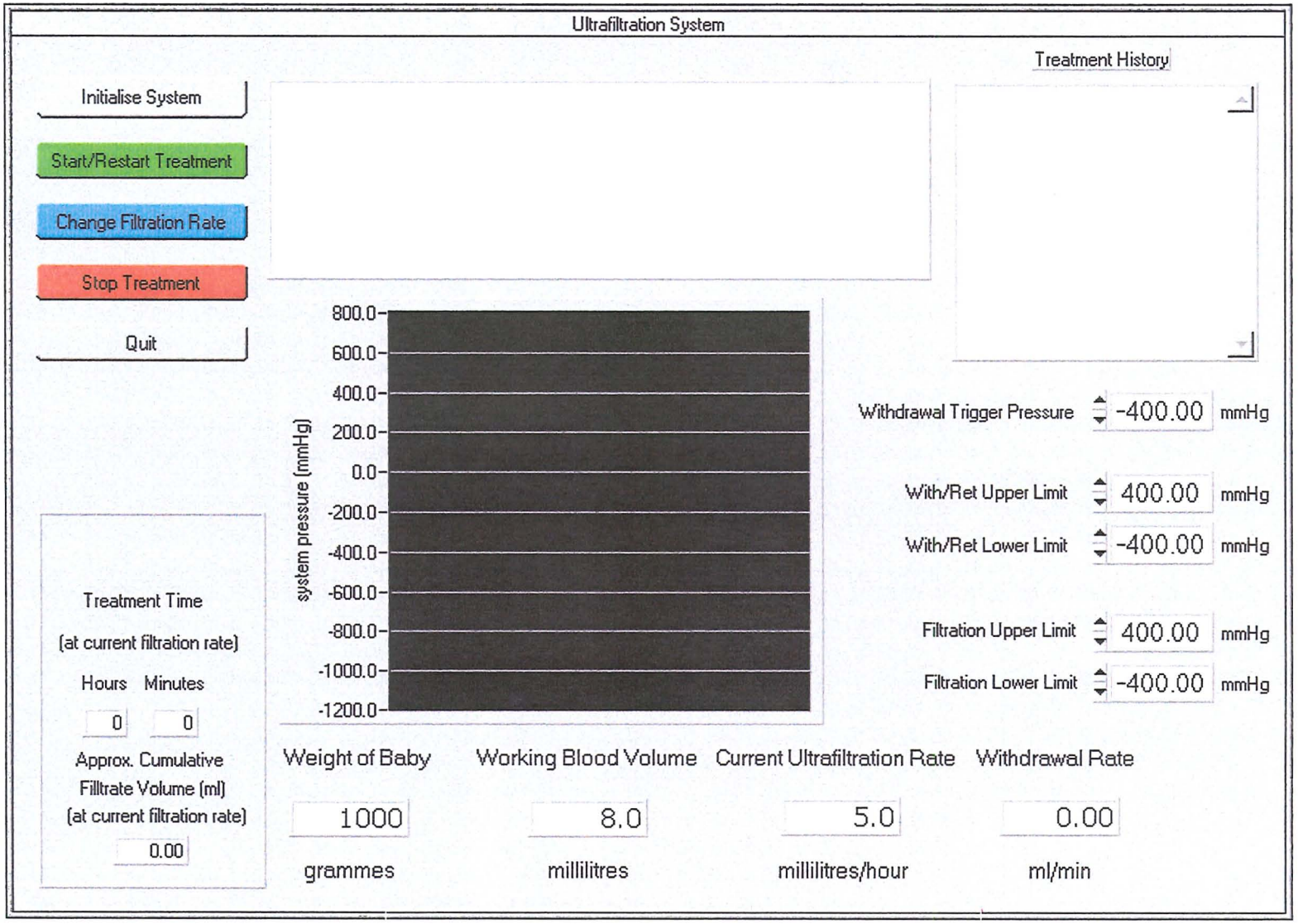


Figure 8.1 Graphical User Interface

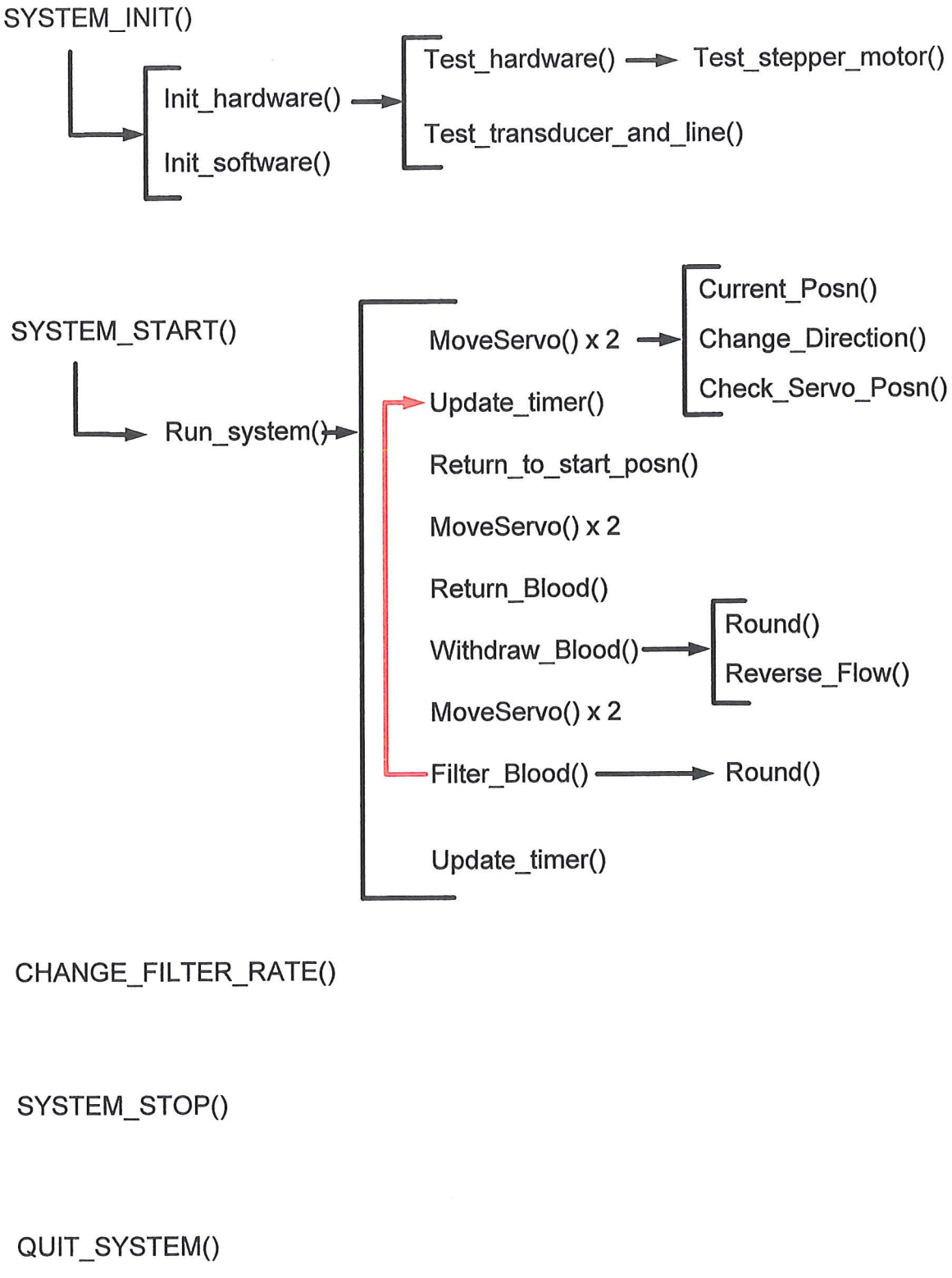


Figure 8.2 Overall Structure of Control Software

Callback functions are in capital letters. The main loop of the operating cycle is indicated by the red arrow.

CHAPTER 9. SYSTEM SAFETY

Introduction

The dialysis system is classified as class I equipment by IEC 60601 [89], the requirements for safety of medical equipment. This is because it has a safety earth and does not rely on double insulation to provide protection against electric shock.

The system has a direct electrical connection to the patient (the venous line). Furthermore, depending on the site of vascular access, the tip of the catheter can be positioned near or actually inside the heart. The patient is therefore susceptible to microshock hazards. A microshock is a very small current flowing through the heart. As little as 10 μA can be sufficient to cause ventricular fibrillation. Because of this, electrical safety requirements are much more exacting than if the system had no direct connection to the patient. In particular, leakage currents flowing between the live and earth terminals of the power supply must be considered, as these are a common source of microshocks.

The electrical safety of the system is discussed in section 9.1. A single fault analysis of the system was undertaken in accordance with the requirements of IEC 60601 parts 1 and 2 [90]. Part 2 addresses the particular requirements for the safety of dialysis machines. This, together with the safety features of the system design, are the subject of the subsequent sections of this chapter.

9.1 Electrical Safety

The entire system (dialysis machine and PC) is connected to the mains electrical supply via a medical grade isolation transformer. Figure E.1 in appendix E illustrates this. The live terminal of the mains supply is at high voltage relative to the earth, which can supply a return current path in the event of an electrical fault. An isolation transformer removes this return current path - the live terminal is only at high voltage with respect to the neutral terminal, not the earth. This greatly reduces the risk of an electric shock to both the patient and the staff operating the system.

The isolation transformer also reduces earth leakage currents. This is particularly important in systems that use a PC. They are usually fitted with power supply suppression circuits, to protect the computer system from fluctuations in the

mains voltage. These include capacitors connecting the live and earth terminals, which allow a large leakage current to flow. An isolation transformer eliminates this source of leakage current.

The system in its current state does not entirely meet the requirements of IEC 60601. Some modifications would be necessary for complete compliance. The switched mode power supply in the dialysis machine should be replaced with a linear supply. This provides a higher degree of isolation. An isolation amplifier should be fitted between the pressure transducer and its interface electronics. This would also improve patient isolation.

9.2 Syringe Drivers

The hardware elements of the syringe driver system are the stepper motors (and their associated electronics) and the microswitch circuits that provide position feedback. The circuit diagrams for these elements were given in figures 6.1, 6.2 and 6.3. The possible failure modes for each element were considered in turn, and an appropriate system response devised so that each fault would result in a safe condition.

9.2.1 *Stepper Motors*

The possible failure modes of the stepper motors are:

- no drive
- continuous drive
- intermittent drive

Obviously, since there are 2 syringe drivers, these failure modes can occur in one or both stepper motors simultaneously. There are several potential hazards associated with syringe driver failure. If one driver fails while the other continues to operate normally, the pressure inside the circuit could rise or fall very rapidly. A pressure rise would lead to haemolysis of red cells and possibly leakage of blood from the circuit. A pressure fall also causes haemolysis and increases the likelihood of air entering the blood circuit. If the syringe drivers travel beyond their limits, the stepper motors are powerful enough to cause permanent damage to the mechanical elements of the system.

There is a degree of protection provided by the design of the stepper motor hardware. Firstly, the clock pulses that are delivered to the motor drive board are generated by the control software, rather than a hardware based pulse generator. This is an inherently safer design. A computer malfunction is very unlikely to result in the motors continuing to operate. If clock pulses are generated in hardware, motor operation independent of computer control is more likely to occur. Secondly, the power cutoff microswitches (figure 6.1) prevent the syringe driver moving beyond preset extremes of travel. If movement occurs beyond the preset limits, the power to the stepper motors is cut, preventing further movement.

Possible failure modes are addressed in the control software. The syringe drivers operate under the control of 5 different functions. These will be dealt with in turn.

Filter_Blood()

As mentioned above, if one or other stepper motor stops running, a rapid rise or fall in pressure will result. The pressure inside the circuit is therefore monitored throughout this function. An initial check is done by lines 36 and 37 (refer to appendix D). Lines 128 and 129 perform a pressure check every time a stepping loop is executed. The pressure in the system is therefore checked 40 times every second. If both stepper motors stop running simultaneously, (e.g. as a result of a power supply failure) this will be detected by line 121 (see page 106 for details). In the event of a failure, program execution is halted, and the function process_error() is called. This sounds an alarm, and displays the details of the error that has occurred on the user interface.

As mentioned earlier, because of the method of clock pulse generation, it is difficult to conceive of a situation where one or both drivers could fail to a state of continuous motion. However, if this did occur it would be detected by the pressure checking algorithm. Intermittent drive to one or both syringes is detected in one of two ways. If only one syringe driver is affected, this will be detected by the rapid rise or fall in pressure that will result. If both drivers are affected equally, this will be detected by line 121 in a similar way to the detection of complete loss of drive - i.e. a step count of 2500 will be reached before the syringe arrives at the end of its stroke.

Return_To_Start_Posn()

This function is very similar to the previous one in terms of failure analysis. The pressure check of lines 248 and 249 detects failure of drive to one or other of the syringes. Line 243 detects the failure of both drivers.

Withdraw_Blood()

This function only controls one syringe driver, so failure modes that involve both drivers simultaneously are not relevant. Failure of the drive is checked initially by lines 77 to 80. Once 20 clock pulses have been issued, the syringe driver should have moved sufficiently for the microswitch to change from closed to open. Obviously this will not happen if the stepper motor has failed. If the stepper motor fails after this point in the function the failure will not be detected. However, this does not represent a hazard. Less blood will be withdrawn than programmed, which is not a dangerous condition. The stepper motor failure will be detected by the subsequent call to the Filter_Blood() function.

Reverse_Flow()

Stepper motor failure is not checked in this function. It is not necessary because this failure mode would not result in a hazardous condition.

Return_Blood()

As with the Withdraw_Blood() function, only one syringe driver is operated by this function. Line 194 checks that the syringe driver is working by monitoring the number of clock pulses that have been issued.

9.2.2 *Microswitches*

The microswitch interface circuit is shown in figure 6.3. There are only 2 possible failure modes here, either logic 1 or logic 0. These are addressed, as before, for each function in turn.

Filter_Blood()

Here the same code at line 121 can theoretically detect failure of both the microswitch (to zero) and the stepper motor, as the logical result of both failures is the same. The number of clock pulses that have been issued is monitored. If this goes above the preset limit the program is terminated. This would happen when the end of the syringe stroke was not detected. In practise, this fault would be caught first by the power cutoff microswitches and pressure checking algorithm. Failure to the 1 state is detected by lines 124 to 126. Every time there is a change of direction of syringe travel, the state of the microswitch on the pulling syringe is tested after 25 steps. This should be at state zero. If it is not, an error is called.

Return_To_Start_Posn()

Failure to zero is checked by line 243. As before, this fault would in practice be dealt with first by other safeguards. A check of failure to the 1 state is not necessary because this cannot result in a hazardous condition.

Withdraw_Blood()

Failure to zero is checked by lines 38 to 40. This function starts with the syringe barrel at the bottom of its travel, so the microswitch should be in state 1. Failure to the 1 state is checked by lines 77 to 80. After 20 steps have been executed, the microswitch should have changed state from 1 to 0.

Reverse_Flow()

There is no explicit code for detecting microswitch failure inside this function, as it is not necessary. A failure to the 1 state will simply result in the stepping loop exiting without a hazardous condition arising. If a failure to the 0 state occurs while the syringe is not near the bottom of its travel the function will execute as normal with no hazard arising. If the syringe is near the bottom of its travel the power cutoff microswitch will operate, resulting in a safe condition.

Return_Blood()

Lines 194 and 195 detect failure to the 0 state. Failure to the 1 state results in the stepping loop terminating so a hazard cannot arise.

9.3 Three Way Tap Drivers

The circuit diagrams for the tap driver system are shown in figures 6.7 and 6.8. Possible modes of failure can be looked at in 2 ways - at the system level and at the level of individual elements of the system. At the system level, the following applies. Each tap can be in one of two positions, either open or shut (see figure 4.2). So there are 2 possible failure modes. When a given tap fails, the other tap can be either open or shut, giving a total of 4 failure permutations. There are 4 basic phases of the operating cycle that are relevant to tap failure - filtration, returning to the start position, blood return and blood withdrawal. This gives a total of 16 failure permutations that need to be addressed. An analysis of these permutations reveals that only 3 represent a potentially hazardous condition. They are:

- tap 2 open, tap 4 shut during withdrawal phase
- tap 2 open, tap 4 shut during return phase
- tap 2 open, tap 4 open during withdrawal phase

Conditions 1 and 3 can result in dialysate being drawn into the blood circuit, and condition 2 would result in an unacceptably high ultrafiltration rate.

These failure modes can also be addressed by looking at the system elements individually. These are the motor drive circuit (figure 6.7), the infrared transmitter circuit and the infrared receiver circuit (figure 6.8). These will be dealt with in turn. The tap driver system is operated by one main function, Move_Servo(). This calls 3 other functions - Current_Posn(), Change_Direction() and Check_Servo_Posn(). The line numbers in the subsequent descriptions refer to the file SERVO2.C in appendix D.

As before, in the event of a failure the following sequence of events takes place. Program execution is halted, process_error() is called, an alarm is sounded and details of the fault are displayed on the user interface.

9.3.1 *Motor Drive Circuit*

An examination of figure 6.7 shows the possible modes of failure here. They are:

- No drive in either direction
- No drive in one direction
- Continuous drive in either direction
- Intermittent drive

No drive in either or both directions is dealt with in the same way. The code in Move_Servo() selects the optimum direction of rotation when searching for the target position. If this has not been found after 1000 pulses have been executed, the motor direction is reversed and the target position is sought in the other direction. If the number of direction changes exceeds 8, an error is called (line 109). So if the motor drive has failed in both directions this error will be called. If only one direction is affected, this error will not be called. The target position will be found, so a hazardous condition will not arise. Depending on the initial position of the tap, the only effect of this fault would be to increase the time taken to find the target position.

There is no specific code to detect an error of continuous drive in either direction. This is not needed because an error such as this would be detected by the pressure checking algorithms inside other functions that are called after the call to the Move_Servo() function. If a 3 way tap is continuously rotating, it would spend the majority of the rotation in a closed position, which would result in an undue build up of pressure, triggering an alarm.

An intermittent loss of drive would be dealt with by the same code that addresses complete loss of drive. Again, either the target position would be achieved (taking a longer time), or the cycle counter would exceed 8 and an error would be called.

9.3.2 *Infrared Transmitters*

These are shown in figure 6.8. This circuit is designed to operate continuously, so the only possible failure mode is a loss of light emission (either complete or partial) by the diodes. This is logically equivalent to a permanent output state of 1 from the phototransistor and also to a loss of drive from the

motors. This is therefore detected and dealt with in the same way as loss of drive, as described in the previous section.

9.3.3 *Infrared Detectors*

Figure 6.8 shows that there are 2 possible modes of failure - to output state 1 or 0. Output state 1 is dealt with as described in section 9.3.1. A failure to 0 is dealt with in 2 different ways, depending on the initial circumstances. Lines 94 to 96 inside the main control loop check that both detector lines go to state 1 during the movement from the initial to the target position. If they do not, lines 111 and 112 call an error indicating a failure. The no_movement flag is needed for the circumstance in which the target position is the same as the current position. In this situation, the state_11 flag will not be set inside the control loop, so another way of testing for failure to 0 is needed. This is provided by lines 117 - 120 and the function Check_Servo_Posn(). The servo is moved a short distance away from the target position until the relevant detector line goes high. It is then moved back to the target position. If the line does not go high an error is called.

9.4 Pressure Transducer Amplifier

The circuit diagram of the amplifier is shown in figure 6.9. The potential failure modes considered were:

- output fails to 0 V
- output fails to 5 V
- output fails to 2.5 V (e.g. pressure transducer diaphragm breakage)

A failure of this system is potentially hazardous. An uncontrolled pressure rise can result in damage to the blood by haemolysis, and blood leakage from the extracorporeal circuit. A pressure drop can result in air entering the circuit with the associated risk of air embolism to the patient. High or low pressures in the venous line can result in damage to the blood vessels of the patient.

Pressure checks are present throughout the control software (see chapter 8 for more details). When the syringe drivers are moving, the pressure inside the circuit is checked on average 40 times every second. Lines 36 and 129 in Filter_Blood() provide an example of the code used. At the start of the function a

reference pressure is read. Each time the control loop executes the current pressure is compared with this reference pressure. Failure of the amplifier output to either 0 V or 5 V would be detected by this code. However, failure due to diaphragm breakage would not be detected. This problem needs to be addressed in future work. A dual system could be implemented with a second pressure detection system providing redundancy. Alternatively, code could be written to monitor the small scale changes in pressure read out to ensure correct pressure transducer function. At present this monitoring is only present in the system initialisation procedure, which will be described in section 9.7.

9.5 Computer System

Ensuring software reliability is a very difficult problem in medical engineering. There is no generally accepted method for assessing and dealing with the risks associated with a system that operates under computer control. This problem will be addressed in the second prototype system by introducing hardware interlocks which provide single fault condition safety independent of the computer control system. The current system provides a degree of protection against software faults. Since the syringe drivers can only move under computer control, a computer crash will always result in a loss of drive to the syringes.

A specific failure mode that is addressed is that of loss of timing control. The filtration phase of operation works on a step cycle that lasts 0.025 seconds. The SyncWait() function (line 112 in Filter_Blood()) provides this timing control. Of itself this function can only ensure that a stepping cycle takes a minimum of 0.025 seconds. If for any reason the code that is executed between 2 calls to this function takes longer than this time, synchronisation will be lost and an inaccurate ultrafiltration rate will be given. This is checked by lines 179 and 180. The total number of steps that have been executed at the end of the filtration phase is checked. If this is less than 9595 then an error is called. Obviously, the software is designed so that the code that has to be executed in the stepping cycle takes less than 0.025 seconds. However, certain errors in the operating system set up can result in the stepping cycle taking longer, so this check is necessary. An example of such an error would arise if the operator had inadvertently changed the display driver settings so that screen updates took longer.

9.6 Extracorporeal Circuit

The extracorporeal circuit is mostly constructed from standard clinical equipment (see figure 4.2). The standard items are obviously already safe for clinical use and as such need no particular safety analysis. The exception is the syringes. The problems that have arisen with these and their safety implications are addressed elsewhere (sections 7.2.1 and 12.2).

The main risk associated with the extracorporeal circuit is failure of one or more of the luer locking joints. This could be as a result of incorrect assembly or the joints working loose during operation. Blood could then leak out or air get into the circuit. The risk of this can be minimised in two ways. The circuit should only be assembled by experienced clinical staff who are familiar with the operating instructions (appendix E). It should also be inspected at regular intervals during operation to make sure all circuit joints are still intact.

9.7 System Initialisation

When the machine is first switched on, it is subject to a thorough test of all systems to make sure they are functioning correctly. This is provided by the software that has already been described in section 8.7. One point is worth reiterating. As part of this testing procedure, the pressure transducer amplifier is checked to make sure it responds to a pressure rise and a pressure drop. So a failure due to a diaphragm breakage (see section 9.4) would be detected by this procedure.

9.8 Safety in Normal Operation

When all elements of the system are functioning correctly, hazardous conditions can still arise which the system needs to be able to deal with. These are detailed below.

Venous Line Occlusion

During the blood return phase the pressure in the extracorporeal circuit is measured continuously. This is done by lines 199 and 200 in the Return_Blood()

function. If the venous line becomes blocked the pressure will rise and trigger the alarm.

The situation during blood withdrawal is complicated by the algorithm used. If the venous line becomes blocked, the syringe will pump back and forth in an attempt to clear the blockage. Therefore, pressure sensing is an inappropriate way to check for a blockage. Line 52 in Withdraw_Blood() sets a time limit on blood withdrawal that is proportional to the volume of blood to be withdrawn. If this time limit is exceeded, as it will be if the line is blocked, lines 142 to 144 will trigger an alarm.

Haemofilter occlusion

Haemofilter occlusion can occur in two ways. There can be a gradual build up of deposits on the inside of the haemofilter fibres, which results in a narrowing of the channels through which blood can flow. The pressure gradient along the haemofilter will therefore increase. This will be reflected in a rising and falling pressure readout from the transducer in time with the reciprocating action of the syringes. If the occlusion becomes bad enough, the pressure changes can increase enough to fall outside the limits set by lines 128 to 130. An alarm will then be triggered.

The other type of occlusion is that caused by thrombus (blood clot) formation in the extracorporeal circuit. This can be caused by insufficient heparinisation of the circuit. The clot will usually lodge at one end of the haemofilter, at the point where the blood flow enters the fibres. The thrombus forms a plug in the fibre ends, as its physical properties prevent the thrombus material from flowing through the fibres. This results in a sudden large change in the pressure readout from the transducer, which will trigger an alarm.

Summary

This chapter has dealt with the important topic of system safety. Although the current system does not meet entirely the requirements of IEC 60601, the work that has been described here goes a long way towards this. The safety shortcomings of the current system will be rectified in the second prototype system. The next chapter discusses the initial testing of the system.

CHAPTER 10. TESTING THE PROTOTYPE SYSTEM

Introduction

Initial testing was undertaken to establish the proper functioning of the various parts of the system. There was particular emphasis on assessing the accuracy of control of ultrafiltration rate. Testing was performed with both water and blood as a working fluid.

10.1 Testing with Water

Once construction was finished, testing could begin. Initial tests were performed using water as the working fluid. The water was held in the barrel of a 60 ml syringe and fed to the circuit. The outflow line from the haemofilter was attached to a 10 ml burette so the output from the system could be accurately measured. The system was run for 1 hour at a time, with various combinations of working fluid volume and ultrafiltration rate. The aim of these tests was to see how accurate the system was with respect to ultrafiltration rate. The results of the tests are given in table 10.1. The differences between the target volumes and the actual volumes achieved are given as percentage errors in the last column. The mean errors and standard errors of the mean for various subcategories of the data and for the data as a whole are given in table 10.2. These figures show that the accuracy achieved was well within the ± 5 % given in the original specification, apart from test 6 in the first test sequence. These results were good enough to allow testing with blood to proceed.

10.2 Testing with Blood

10.2.1 *Redesign of Three Way Tap Drivers*

The initial volume tests using blood as the working fluid were unsuccessful. Contamination with blood increased greatly the torque needed to turn the three way taps, and the servos driving them stalled. This problem was solved as detailed in section 7.2.2.

10.2.2 *Blood Stirrer*

During the testing the donated blood is supplied from a reservoir consisting of the barrel of a 60 ml syringe. Over a period of time the formed elements in the blood settle to the bottom of the reservoir, forming a thick sludge. This results in blood of a very high haematocrit being supplied to the system, giving false test results and increasing the chance of a blockage in the haemofilter. This problem was solved by the introduction of a blood stirring system. The circuit diagram for this is shown in figure 10.1. It is a simple power transistor switch. The aluminium stirrer is placed into the top of the syringe barrel. It is controlled by line 30 of the DAQ board so that the motor can be switched on only during the blood return phase of the operating cycle. This minimises the breakdown of blood cells that inevitably occurs with any sort of mechanical stirring. The digital line is connected through 2 switches. S1 allows the stirrer to be completely disconnected from the DAQ card. S2 allows a short period of stirring to be done manually. A 7404 inverter delivers the logic signal to the base of the transistor via a 500 Ω current limiting resistor. Originally the motors were driven directly by the power transistor. However this proved to be unreliable, as the motors would sometimes stall when switched on, due to the slow rise time in the voltage delivered by the transistor. This problem was solved by driving the motors via a relay and contacts S3. The 7.8 Ω resistance limits the current delivered to the motors, thereby reducing their speed. The motors themselves are 3 V model types, and the gear ratio used is 1:16. The slow rotation of the stirrer reduces haemolysis (cell breakdown) of the blood.

The mechanical components can be seen in figure 10.2. 2 motors are needed to provide a control for the clinical testing (see chapter 11).

10.2.3 *Method*

A series of tests were undertaken using donated blood. 200 ml was available for the tests. The blood was heparinised as soon as it was taken, at a concentration of 10 units of heparin per ml of blood. It was stored in a refrigerator until needed. Adding heparin prevents the blood coagulating, and refrigeration slows the breakdown of the cells. Typically 40 ml would be used for each test, held in the barrel of a 60 ml syringe. This syringe was connected to the machine using

three way taps and luer ended tubing. The filtrate was collected in a 10 ml burette so that the volumes obtained could be accurately measured. It is important to keep the haematocrit of the blood constant during a test run. As filtrate is extracted, the overall volume of the blood is reduced, while the volume of the formed elements (the cells) remains constant. This results in an increase in the haematocrit and therefore a reduction in the ultrafiltration rate. So as filtrate is withdrawn from the blood, it is replaced by heparinised normal saline to keep the total volume of the sample constant.

Most test runs lasted for 1 hour. Different combinations of working blood volume and filtration rate were tested. In all 18 successful test runs were made. The ultrafiltration rates achieved were recorded.

10.2.4 *Results*

The results are shown in table 10.3 . It was found that the rates were all consistently lower than the rates obtained with water, as expected. The analysis of the errors is shown in table 10.4. The error was not a function of the target filtration rate (see figure 10.3), so it was possible to calculate a correction factor that could be applied to the filtration rate algorithm to bring the volumes back within the specification. This was done as follows. A mean percentage error was calculated for all the tests that had a working blood volume of 5 ml. The result of test 26 was ignored as the error here was nearly 10 % greater than for any other test.

The mean error was found to be -9.05 %. So the correction factor could be calculated:

$$\text{Correction Factor} = \frac{100}{100 - 9.4} = 1.104$$

Once this correction factor is included in the program, the filtration rate accuracy becomes, at ± 4.5 %, better than the ± 5 % required by the specification. These figures compare well with the accuracy obtained in conventional haemodialysis circuits. The volumetric pumps used to control ultrafiltration flow can have an error of as much as 10 % between the set flow rate and the actual flow rate delivered. Roberts and Winney [81] report overall errors of as much as 5.8 %, and Jenkins et al. [82] report a maximum error of 12.5 %.

It is worth noting that most methods of haemodialysis in small babies do not allow direct control over ultrafiltration rate. An extracorporeal circuit that relies on the patient's blood pressure to supply the transmembrane pressure for ultrafiltration will be limited in the amount of ultrafiltration that it can achieve. The syringe driven system however, can control the ultrafiltration rate directly and does not rely on the patient's mean arterial blood pressure to supply the driving force for ultrafiltration. This is a significant advantage over conventional methods of treatment. The maximum ultrafiltration rate that can be achieved for a given weight of baby can be estimated as follows:

Assume that each volume of blood that is withdrawn into the system can be safely reduced to 60 % of its original volume by ultrafiltration. Any further reduction would result in a very large increase in haematocrit and therefore increase the risk of the blood clotting in the circuit.

So, $\Delta Q_F = 0.4\ V_B$

where ΔQ_F = amount of ultrafiltrate produced in one operating cycle

and V_B = working blood volume

The time taken for 1 operating cycle can be worked out assuming a blood flow rate of 10 ml / min:

cycle time, Δt = filtration time + withdrawal time + return time

$$= 4 + \frac{V_B}{10} + \frac{(1-0.4)V_B}{10} \qquad (10.1)$$

So, maximum ultrafiltration rate, $Q_F = \frac{\Delta Q_F}{\Delta t} = \frac{4V_B}{40 + 1.6V_B}$ ml / min (10.2)

The working blood volume can be related to the weight of the baby using the standard formula that was employed in the user interface (see section 8.7). This assumes a blood volume of 85 ml per kg of body weight, and that it is safe to remove 10 % of this.

So, $V_B = 0.1 \times (85 \times W) = 8.5\ W$ ml (10.3)

where W = weight of baby in kg

Substituting this in equation 10.2 gives

$$Q_F = \frac{34\ W}{40 + 13.6\ W} \text{ ml / min} \qquad (10.4)$$

The performance of the syringe driven system can now be compared with other methods of treatment. Zobel et al. [27] describe 6 treatments using continuous arteriovenous haemodialysis. They quote average values of ultrafiltration rates of 2.1 ml/min/m^2 for CAVH and 1.7 ml/min/m^2 for CAVHD. These figures are given as ultrafiltration rate per unit surface area of the baby, which is a common practise in paediatric nephrology. The mean patient weight is quoted as 3.2 kg, so the figures can be converted using the usual methods that relate body weight to surface area [21]. This gives values of 0.47 ml/min and 0.38 ml/min. Using the value 3.2 kg in equation 10.4 gives a maximum filtration rate of 1.3 ml/min, which is nearly 3 times higher than the value quoted for CAVH. Lieberman [32] reports a UF rate of only 2.4 ml/h for 1.3 kg baby using CAVH.

The relevance of models of ultrafiltration such as those described by Pallone [48] was considered. These models were discussed in section 2.4. This type of analysis applies to extracorporeal circuits employing continuous arteriovenous haemofiltration. The volume flow rate of ultrafiltrate is usually the dependent variable, and is a function of parameters such as transmembrane pressure and haematocrit etc. In some circuits a volumetric pump is attached to the ultrafiltrate line, so ultrafiltrate flow rate becomes the independent variable and transmembrane pressure the dependent variable. This is analogous to the syringe driven system. Direct control over the volume of the extracorporeal circuit means that ultrafiltration is an independent variable, and any model of this sort would predict the transmembrane pressure inside the haemofilter. However, this variable is not in itself of much interest if the ultrafiltration rate is already determined. This assumes that the transmembrane pressure remains well inside the safety limits for the haemofilter, which it does in normal operation. The upper limit quoted for the Miniflow 10 is 450 mm Hg [87]. The safety systems built into the machine would halt operation long before the pressure got this high.

The other obvious problem with these models is that they apply to steady state conditions in the extracorporeal circuit, such as are found in continuous arteriovenous haemofiltration. This is clearly not the case for the syringe driven system. Not only are there 3 distinct phases during the cycle of operation (blood withdrawal, filtration and return), but also the direction of blood flow through the haemofilter reverses several times during the filtration phase. It would therefore seem that steady state models would not provide useful predictions of the performance of the syringe driven system.

Summary

The early testing confirmed that the system was operating satisfactorily from a technical point of view. In particular, the ultrafiltration rate control was within ± 5 %, a major requirement of the specification. Once these facts had been established, it was necessary to move on to the clinical testing of the system. This is the subject of the next chapter.

Date	Test No.	UF Rate ml/h	Working Fluid Volume ml	Time min	Calculated Volume ml	Actual Volume ml	Percentage Error
29/04/98	1	15	10	62.97	15.57	15.5	-0.45
	2	10	10	63.16	10.53	10.5	-0.28
	3	3	10	64.12	3.18	3.3	3.77
	4	15	3	62.96	15.58	15.7	0.77
	5	10	3	64.51	10.74	11	2.42
	6	3	3	64.66	3.19	3.4	6.58
	7	15	5	61.3	15.32	15.2	-0.78
	8	10	5	65.22	10.82	10.6	-2.03
	9	3	5	64.41	3.18	3.2	0.63
	10	3	5	65.12	3.22	3.2	-0.62
07/09/98	1	10	5	60.56	10.11	10	-1.09
	2	3	5	64.91	3.23	3.2	-0.93
	3	15	5	56	14.17	13.95	-1.55
15/01/99	1	5	5	33	2.81	2.8	-0.36
	2	3	5	59	2.98	3.1	4.03
	3	10	5	61.4	10.17	10	-1.67
	4	15	5	45	11.45	11.25	-1.75
	5	15	5	62.12	15.62	15.5	-0.77

Table 10.1 Water Test Results

Category	Mean Error	SEM
3 ml/h, 5 ml	0.776	1.135
10 ml/h, 5 ml	-1.60	0.275
15 ml/h, 5 ml	-1.21	0.255
All Data	-0.04	0.444

Table 10.2 Analysis of Water Test Data

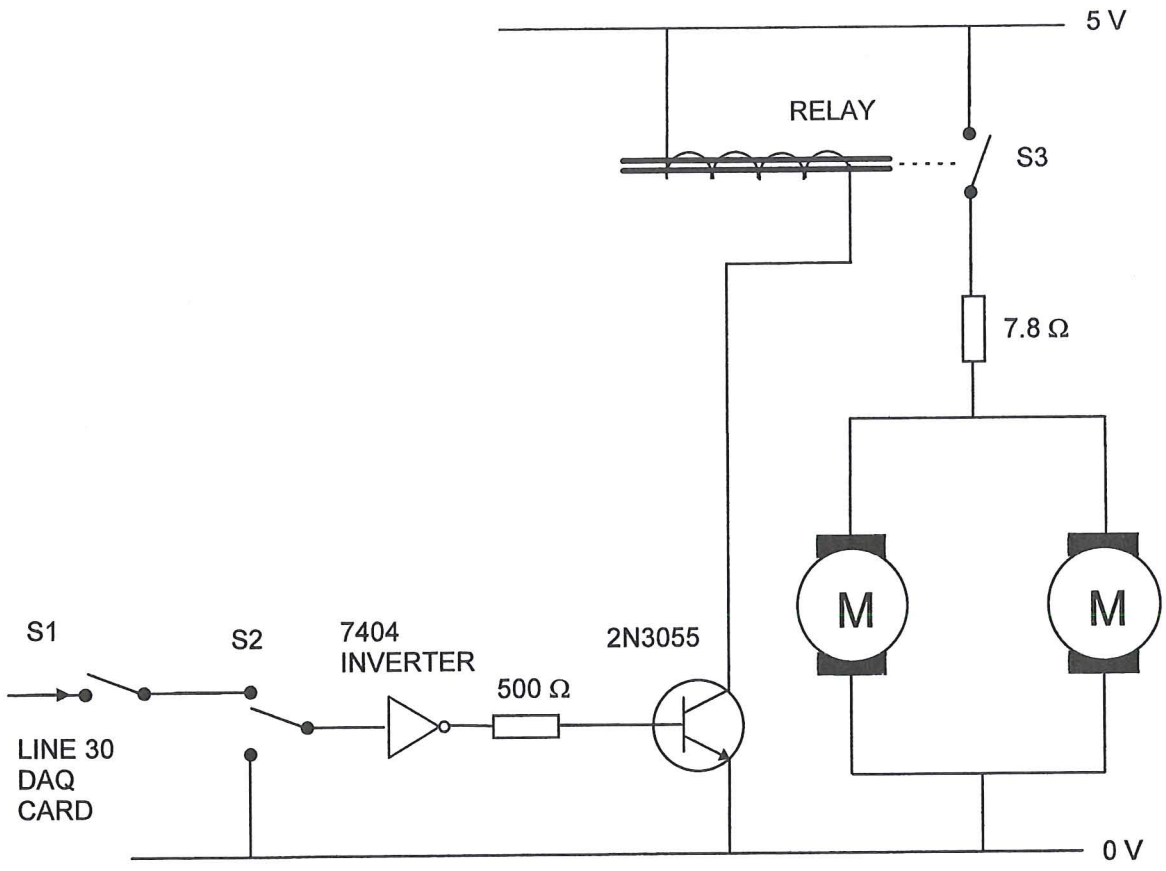


Figure 10.1 Circuit Diagram of Blood Stirrer

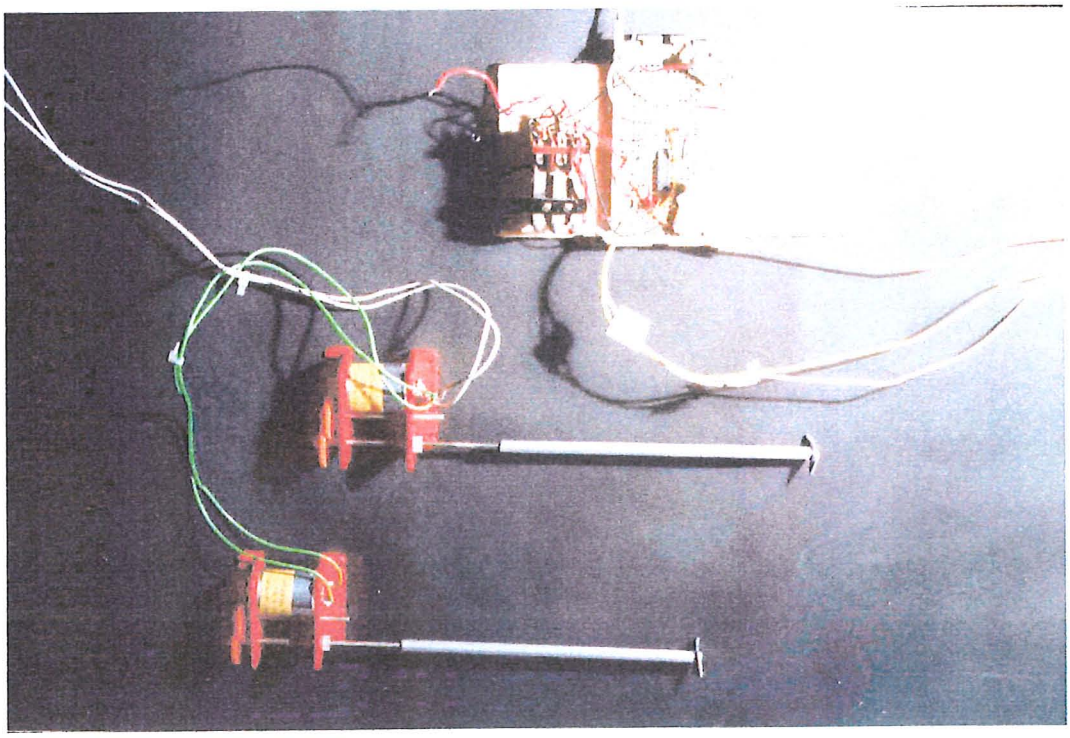


Figure 10.2 Blood Stirrers

date	test no.	UF rate ml/h	working fluid volume ml	time min	calculated volume ml	actual volume ml	percentage error
07/09/98	4	3	5	60	3.03	2.95	-2.64
	5	10	5	62.72	10.39	9.5	-8.57
	8	3	5	123.33	6.13	5.6	-8.65
	9	15	5	61.76	15.29	13.9	-9.09
	10	15	10	66.47	16.49	15.75	-4.49
	11	3	5	60.96	3.03	2.8	-7.59
	12	5	5	61.37	5.06	4.75	-6.13
	13	7	5	65.37	7.59	6.8	-10.41
	14	9	5	63.13	9.46	8.7	-8.03
	15	11	5	63.04	11.48	10.15	-11.59
	16	13	5	58.03	11.7	10.5	-10.26
	17	3	5	60.14	3	2.6	-13.33
	18	5	5	61.65	5.1	4.35	-14.71
	19	7	5	62.19	7.18	6.5	-9.47
	22	5	3	60.85	5.07	4.4	-13.21
	23	5	6	60.76	5.06	4.65	-8.10
	24	5	8	62.15	5.18	4.85	-6.37
	25	5	10	61.51	5.12	4.75	-7.23
	26	9	5	63.54	9.48	7.7	-18.78
	27	11	5	64.7	10.79	9.65	-10.57
	28	15	5	60.15	11.52	10.5	-8.85
	29	13	5	65.19	9.81	8.75	-10.81

Table 10.3 Ultrafiltration with Blood as Test Fluid

Category	Mean Error	SEM
3 ml/h	-8.05	2.19
5 ml/h	-9.29	1.52
15 ml/h	-7.48	1.50
All Data	-9.05	0.63

Table 10.4 Analysis of Blood Test Data

CHAPTER 11. CLINICAL TESTING

Introduction

Once the basic functions of the system had been tested and found to work satisfactorily, the clinical testing could begin. Initially two aspects of the system's operation were examined. The first of these was a study of the clearances that could be achieved by adding dialysis to the system. This is important because dialysis is often preferred over ultrafiltration as the choice of treatment. Sometimes both are used in combination. Dialysis can provide greater effective clearances.

The second was an investigation of the amount of haemolysis (cell breakdown) that occurs inside the system. Haemolysis results from mechanical damage to the blood cells. This will occur particularly at the point where the syringe plunger contacts the inside of the barrel.

Most of the tests were performed twice, once for the 10 ml syringe system and once for the redesigned 25 ml syringe system.

11.1 Addition of Dialysis to the System

The system was modified to provide dialysis as well as ultrafiltration with the addition of a few pieces of standard clinical equipment. A 5 litre bag of Hospal Hemosol dialysate fluid was used. This system is ideal for paediatric use as it is much simpler to prepare than the fluids used for adult dialysis. The relatively small amount of fluid in each preparation is not a drawback as the fluid flow rates needed to dialyse a premature baby are very small, of the order of 70 ml/h. The dialysate is delivered to the haemofilter via a cartridge type intravenous pump. Such a pump can deliver an accurately monitored flow rate through the haemofilter. The pump is connected to the inflow dialysate port of the haemofilter, and a waste tube is connected to the outflow port.

11.2 Dialysis Testing

Testing was done on both the 10 ml syringe system and the 25 ml syringe system. Very similar procedures were used for both systems.

There were 2 main parts to the investigation when the 10 ml syringe system

was tested. The first was to look at the filtration time. The original specification called for 4 minutes of blood filtering for each operating cycle. This was based on clinical experience. However, it was not known if this was the optimum filtration time, so it was decided to test different times and compare the results. The times used were 1 minute, 2 minutes and 4 minutes. Other parameters (i.e. blood withdrawal and return times) were kept the same between the 3 tests. The working blood volume was set at 3 ml and the dialysate flow rate at 100 ml/h.

The second part was to measure clearance rates, to make sure the system could achieve the clearances that are necessary in clinical practice. 2 parameters were varied in this investigation, the working blood volume and the dialysate flow rate. It was hoped to estimate the optimum dialysate flow rates. A balance needs to be struck between maximising clearances and minimising the dialysate flow rate, so as not to be wasteful in the use of dialysis fluid.

8 tests were done in all, with the parameters shown in table 11.1 below.

Working Blood Volume (ml)	Dialysate flow rate (ml/h)
3	30
3	60
5	50
5	100
5	600
7	70
7	140
10	100
10	200

Table 11.1 Test Parameters

The dialysate flow rates were chosen by first calculating the effective blood flow rate from the reservoir (or the patient in the clinical situation). This is much lower than the blood flow rate through the haemofilter itself, which is 10 ml/min. This is because the same blood flows through the haemofilter several times in each operating cycle. An example calculation follows:

If the working blood volume is 5 ml, and the operating cycle takes 333 seconds, then the effective blood flow rate from the patient is:

$$\text{flow rate} = \frac{5}{333} = 0.015 \text{ ml/s} = 54 \text{ ml/h}$$

So the first dialysate flow rate was chosen to be roughly the same as the effective blood flow rate, and the second to be double this rate. Additionally, for a working blood volume of 5 ml, a 600 ml/h rate was included. This is the same as the actual blood flow rate through the haemofilter. It was hoped to test if a greatly increased dialysate flow rate had a significant effect on the clearance rate.

The above tests (not including the filtration time experiment) were repeated for the 25 ml syringe system. The only difference was in the range of working blood volumes used. Since the syringes were bigger a maximum working blood volume of 20 ml was used.

11.2.1 *Method*

The clinical apparatus was modified as shown in figure 11.1 to allow sampling of blood and dialysis fluid to take place. At the start of each test the IV pump is switched on to start the dialysate pumping through the haemofilter. Tap C is in position 1 so that the outflow from the haemofilter can flow into the waste reservoir. Tap A is in position 1 to allow undialysed blood to be drawn into the circuit from the reservoir. As in previous tests, the blood reservoir is stirred regularly to prevent the formed elements settling to the bottom. At the end of the filtration phase tap A is turned to position 2 to divert the dialysed blood into a waste blood collector via tap B which is in position 1. This cycle of operation is repeated 3 times to allow the biochemistry of the fluids to settle to a steady concentration. At the end of the third cycle, tap B is turned to position 2 so that the dialysed blood can be sampled, being collected in the 10 ml syringe attached to tap B. Tap C is turned to position 2 at the beginning of the third cycle so that the waste dialysis fluid is sampled over one complete operating cycle.

To increase the accuracy of the biochemical assays, it is necessary to artificially increase the levels of creatinine and urea in the donated blood, taking them up to typical values that would be found in a patient with acute renal failure. For creatinine, the normal adult concentration range is 45 - 120 μmol/l. It was

decided to increase the concentration by 320 μmol/l. The total volume of blood available for the tests was 200 ml. So the mass of creatinine that must be added can be calculated:

Creatinine does not enter the blood cells, so for the calculation only the plasma volume needs to be considered. Assuming a haematocrit of 50 %, the plasma volume is 100 ml.

$$\begin{aligned} \text{No. of moles of creatinine} &= \text{concentration} \times \text{volume} \\ &= 320\ \mu\text{mol/l} \times 100\ \text{ml} \\ &= 32\ \mu\text{mol} \end{aligned} \qquad (11.1)$$

$$\begin{aligned} \Rightarrow \text{Mass of creatinine} &= \text{no. of moles} \times \text{molecular weight} \\ &= 0.032\ \text{mmol} \times 113 \\ &= 3.6\ \text{mg} \end{aligned} \qquad (11.2)$$

For urea, the normal adult range is 3.3 - 6.7 mmol/l. This was to be increased by 18 mmol/l. Urea does enter the blood cells, so the whole volume of 200 ml needs to be considered for the calculation:

$$\begin{aligned} \text{No. of moles of urea} &= \text{concentration} \times \text{volume} \\ &= 18\ \text{mmol/l} \times 200\ \text{ml} \\ &= 3.6\ \text{mmol} \end{aligned} \qquad (11.3)$$

$$\begin{aligned} \Rightarrow \text{Mass of urea} &= \text{no. of moles} \times \text{molecular weight} \\ &= 3.6\ \text{mmol} \times 40 \\ &= 144\ \text{mg} \end{aligned} \qquad (11.4)$$

So, 3.6 mg of creatinine and 144 mg of urea were added to the blood sample and mixed in well. The blood was left to stand for an hour before testing began, to allow the urea to be absorbed into the cells.

As in previous tests, the blood was heparinised at a concentration of 10 units/ml.

One 2.5 ml blood sample was taken at the start to allow a full blood count to be performed. This was necessary so that the packed cell volume of the blood sample could be determined. This is needed for the clearance calculations (see

section 11.2.2).

Two control samples were also taken, one from the blood and one from the dialysis fluid.

The testing could then begin. 12 blood samples and 12 fluid samples were taken according to the regime outlined above. They were collected in standard clinical 2.5 ml sample tubes. Since the test was designed to determine the clearances that could be achieved by dialysis alone, the system ultrafiltration rate was set to zero.

11.2.2 *Results and Conclusions*

Results

The blood and fluid samples for the 10 ml syringe system tests were assayed to determine the concentrations of potassium, urea and creatinine. One sample underwent a full blood count to determine the PCV (packed cell volume). The control results were as follows:

Blood Control

PCV: 0.439

Potassium: 3.8 mmol/l

Urea: 22.1 mmol/l

Creatinine: 223 μmol/l

Fluid Control

Potassium: 0.1 mmol/l

Urea: 0.0 mmol/l

Creatinine: 0 μmol/l

The test sample results are given in columns 5 to 10 of table 11.2. For each test, the clearance can be calculated for each of the concentrations tested, so 6 values for clearance can be obtained. Clearances are calculated as follows.

As was explained earlier, plasma clearance is defined as the volume of plasma that is completely cleared of a given solute per unit time. So, the formula for clearance in terms of plasma concentrations is:

$$C = Q_P \times \left(\frac{P_I - P_O}{P_I} \right) \qquad (11.5)$$

where clearance is in ml / min and

Q_P = plasma flow rate (ml / min)

P_I = plasma concentration in (mmol / l or μmol / l)

P_O = plasma concentration out (mmol / l or μmol / l)

The effective blood flow rate through the system is found by dividing the working blood volume by the operating cycle time (columns 2 and 4). However, since it is the plasma clearances that must be calculated, the effective plasma flow rate is needed. This is the blood flow rate multiplied by the volume fraction of plasma in the blood. So the formula for Q_p is:

$$Q_P = \frac{V_{WB} \times (1 - PCV)}{t_c} \qquad (11.6)$$

where V_{WB} = working blood volume (ml)

PCV = packed cell volume (l / l)

t_c = cycle time (min)

Combining the two formulae gives a single formula that can be used in the spreadsheet to calculate clearances:

$$C = \left(\frac{P_I - P_o}{P_I} \right) \times \left(\frac{V_{WB} \times (1 - PCV)}{t_c} \right) \qquad (11.7)$$

The relevant control blood concentration is used for P_I, and values are taken from columns 5, 6 and 7 for P_O . V_{WB} is taken from column 2 and t_c is taken from column 4. The value for PCV is 0.439.

A similar formula is used to calculate clearances based on dialysate fluid concentrations. It is based on the principle that the rate of addition of a given solute to the fluid must equal its rate of removal from the plasma. So, the formula is:

$$C = \left(\frac{F_O - F_I}{P_I}\right) \times \frac{Q_F}{60} \qquad (11.8)$$

where F_O = fluid concentration out (mmol / l or μmol / l)

F_I = fluid concentration in (mmol / l or μmol / l)

P_I = plasma concentration in (mmol / l or μmol / l)

Q_F = dialysate fluid flow rate (ml / h)

F_O is taken from columns 8, 9 and 10. F_I is the relevant fluid control concentration. P_I is again the relevant blood control concentration. Q_F is taken from column 3.

With these two formulae the clearances in columns 11 to 16 can be calculated. Finally, the means of all 6 clearances were calculated (column 17).

The same procedure was followed in the 25 ml syringe system tests. The control results were as follows:

Blood Control

PCV: 0.305

Potassium: 17.6 mmol/l

Urea: 15.4 mmol/l

Creatinine: 240 μmol/l

Fluid Control

Potassium: 0.5 mmol/l

Urea: < 0.2 mmol/l

Creatinine: 1 μmol/l

It is interesting to note that the potassium levels here were much higher than in the first set of tests (3.8 mmol/l for the 10 ml syringe tests). This is because bank blood was used for these tests whereas fresh donor blood was used for the first set of tests. A certain amount of haemolysis inevitably occurs in bank blood that has been stored for any length of time. Haemolysis results in potassium being released from inside the red blood cells into the plasma, thus raising the concentration present in the plasma.

The test sample results for the 25 ml system are shown in table 11.3.

Conclusions

The clearances for the 10 ml syringe tests are shown graphically in figures 11.2, 11.3, 11.4 and 11.5. Those for the 25 ml syringe tests are shown in figures 11.6, 11.7 and 11.8. Theoretically, the clearances calculated using plasma concentrations should be the same as those calculated using dialysate concentrations (for a given solute). The results clearly show that this is not the case for these tests. It is hard to say whether this is as a result of inaccuracies in the test procedure itself or in the assays that were performed. However, the results were sufficient to show that the clearances are high enough for the system to be of practical use clinically. It is clear from the graphs that clearance is approximately a linear function of effective blood flow rate over the range of flow rates that were tested. The variation within the test sequences was as expected, with 2 exceptions in the 10 ml series. The plasma creatinine clearances for tests 1 to 3 were inconsistent, as were the fluid potassium clearances for tests 6 to 8.

The filtration time tests (figure 11.5) showed that there is a significant increase in clearance as the filtration time is reduced. A 2 minute filtration phase gives a 20 % increase in clearance over a 4 minute phase. And a 1 minute phase gives a 37 % increase. However, this test does not take into account the problem of blood access. These tests were all done with the same blood withdrawal rate of 10 ml/min. If blood access to the patient is good and a high withdrawal rate can be achieved, it makes sense to reduce the filtration time. However, if blood access is bad, the blood withdrawal phase may take a great deal longer, significantly reducing the mean blood flow rate from the patient. In this case, it is preferable to increase the filtration phase time, to achieve a better clearance. It should be possible to arrive at an optimum filtration time for a given blood withdrawal rate that maximises clearance values.

As expected, the dialysate flow rate tests showed that an increase in flow rate does produce an increase in clearance. Tests 6 to 8 in the 10 ml series provide an example. A dialysate flow rate of 50 ml/h produces a clearance of 0.268 ml/min. Here the dialysate flow rate is similar to the blood flow rate, which in this case is 60 ml/h. Doubling the dialysate rate produces a 20 % increase in clearance. Increasing it to 600 ml/h gives a 36 % increase. While these increases are significant, they are not large enough to justify the increased rate of use of dialysate.

The results for the 25 ml syringe tests were very similar to those obtained

for the tests of the 10 ml syringe system. Obviously, higher clearances were obtained for the higher working blood volumes, as was expected.

It is interesting to compare the test results with clearance figures obtained with conventional methods of dialysis. Zobel, Kuttnig and Ring [27] report obtaining a urea clearance of 6.6 ml/min/m^2 for a body weight of 3.2 kg (values are the mean over 6 treatments) using CAVHD. By the same method as was used in section 10.2.4 this figure can be converted to units of ml/min. A figure of 1.48 ml/min is obtained. For the syringe driven system, a baby with a body weight of 3.2 kg could have an effective blood flow rate through the system of 2.5 ml/min. Extrapolating from figure 11.7, the urea clearance is predicted to be 1.04 ml/min. This is approximately 70 % of the figure obtained by Zobel et al.. At first sight the syringe driven system might be expected to deliver considerably lower clearances than a CAVHD system. This is because the latter utilises a continuous countercurrent flow of dialysate, whereas the syringe driven system alternates between countercurrent and concurrent flow. However, the relatively large dialysate flow rates as compared to the flow rate of blood through the system can be expected to reduce this disadvantage to a minimum. So the clearances obtainable should approach those obtained with CAVHD.

Clinical continuous haemodialysis aims to achieve a clearance between 10 and 20 % of the glomerular filtration rate that the healthy kidneys would achieve. So in assessing the performance of the syringe driven system it is necessary to compare the clearances obtained with the GFR that a healthy baby of the same body weight would be producing. Empirical relationships are available which allow the GFR to be estimated [91]. Using these relationships, the GFR for a 3.2 kg baby can be estimated to be 3.9 ml/min. The figure of 1.04 ml/min obtained above is therefore 27 % of the GFR of a healthy baby of this weight, which is considerably better than the performance needed for the system to be clinically useful. The estimated GFRs for a range of birthweights are compared to the clearances obtainable by syringe driven haemodialysis with the 25 ml syringe in figure 11.9 (dialysate flow rates are 20 times the blood flow rate). It is interesting to note that at weights of 0.8 kg and below the GFR and the clearance are very similar. Above this weight, the GFR increases more rapidly than the available clearance. However, at the maximum weight of 2.35 kg, the clearance is still 46 % of the GFR, which is much higher than the clinically acceptable level.

Various models of clearance (see section 2.4) were considered to see if they were relevant to the experimental data obtained from the syringe driven

haemodialysis system. These models predict clearance under conditions of steady state countercurrent flow - this is the flow regime that is found in conventional haemodialysis systems. The syringe driven system alternates between countercurrent and concurrent flow during the filtration phase of its cycle, therefore flow conditions are unsteady. Because of this, steady state models cannot provide accurate predictions of clearance. Nevertheless, it is still instructive to compare the experimental data with the predictions of such models.

The model of Sargent and Gotch [53] predicts clearance as a function of blood and dialysate flow rates, membrane surface area and mass transfer coefficient K_0, (see section 2.4). It assumes a linear change in solute concentration along the haemofilter on both the blood and the dialysate side. The membrane surface area for the Miniflow 10 [87] is 0.042 m^2. Data for blood and dialysate flow rates was taken from the 25 ml syringe tests (the first 6 rows of data in table 11.3 were used). K_0 cannot be determined directly [54]- its value is obtained by curve fitting of the predicted clearances to the experimental data. Pallone [56] has obtained a value of 0.015 cm/min for the AN69 membrane, which is the membrane used in the Miniflow 10. Sargent assumes that the value of K_0 is constant, whereas other authors [59] conclude that it varies with dialysate flow rate. These values were put into equation 2.18 to obtain the graph shown in figure 11.10 (blue line). The predicted clearance is approximately twice the experimentally obtained clearance. The syringe driven system alternates between countercurrent and concurrent flow, whereas the model assumes a continuous countercurrent flow of dialysate. Therefore it is to be expected that the model would predict higher clearances than those actually obtained.

The model of Pallone et al. [56] derives an exponential relationship between solute concentration and distance along the haemofilter. This model addresses haemodialysis specifically, assuming that no ultrafiltration takes place. It includes another variable - the fractional volume of blood into which urea can distribute. This in turn is a function of protein concentration (C_p) and haematocrit (H) :

$$f = (1-H)(1-0.0107C_p) + 0.86\ H \qquad (11.9)$$

For the purposes of this comparison, the protein concentration was assumed to be zero. This is a reasonable assumption because the blood used in the experiment was derived from packed red cells that had been diluted with normal saline. The

haematocrit in this experiment was 0.305. This figure in equation 11.9 gives a value of 0.957 for f. Using the same value of K_0 (0.015 cm/min) as previously, the predicted clearances are as shown in figure 11.10 (red line). Again, the predicted values are higher than the experimental ones. Since this model also assumes a continuous countercurrent flow, the explanation for the discrepancy is the same as for the Sargent model.

An analysis by Akcahuseyin et al. [55] was also considered. This is a model of continuous arterio-venous haemodiafiltration (CAVHD), which involves both convective and diffusive transfer of solute. The analysis does not extend to conditions of zero ultrafiltration, so it could not be applied to this experimental data. Another analysis by Jaffrin et al.[92] contained too many errors to be useful for comparison purposes.

An unsteady model of the syringe driven system is needed to provide more accurate predictions of clearance. The data obtained from the initial testing of the system is insufficient to produce such a model. Further instrumentation of the extracorporeal circuit and more complicated methods of fluid sampling are needed to provide comprehensive experimental data on which the model could be based.

11.3 Haemolysis Testing

11.3.1 *Method*

The purpose of this test was to determine the amount of cell breakdown being caused by the action of the dialysis machine itself, over a 12 hour period. With the experimental set up being used, there are three main sources of haemolysis. The first is the action of the machine itself. The second is due to the blood in the test reservoir being agitated by a mechanical stirrer. The third comes from the natural breakdown of red cells in the blood sample. To eliminate the last two factors it was necessary to set up a controlled experiment. This was done as follows.

A sample of 100 ml of donated fresh blood was heparinised at 5 units/ml. This was then divided equally between two 60 ml syringe barrels, giving one control sample and one test sample. As before, the barrel which was supplying blood to the machine had to be stirred periodically to prevent the formed elements in the blood from settling to the bottom of the reservoir. However, this time two

stirrers were used, one in the test reservoir, and the other in the control reservoir. They were wired in parallel to the same drive circuit. As before, the stirrers would only be switched on during the blood return phase of the operating cycle. In this way, the control blood and the experimental blood would receive exactly the same amount of agitation from the stirrers, and therefore the same amount of haemolysis from this source.

The machine was run continuously for 12 hours at a zero ultrafiltration rate. Every hour a 2.5 ml sample was taken from the top of each of the reservoirs. This was stored in an EDTA sample bottle. EDTA is an anticoagulant that prevents the blood from clotting. The control and test samples were kept at the same temperature so that the same amount of natural haemolysis would occur in each.

11.3.2 *Results and Conclusions*

The 24 samples were tested for free haemoglobin. The results for the 10 ml syringe system are shown in table 11.4, and in graphical form in figure 11.11. The graph shows that the free haemoglobin in the control sample rose steadily, in a reasonably linear fashion, throughout the 12 hours of the test. The test sample shows a steeper slope, as expected, but the data points are much less consistent, which was not expected. This plot was expected to show a steady rise in free haemoglobin levels over and above the control levels. However the values show marked rises and falls, especially after 6 hours. It is unlikely that there were any errors introduced in the sampling technique, so it is more likely that these variations are due to errors in the free haemoglobin measurement. For both the control and test samples, it is not possible for the free haemoglobin levels to actually fall during the experiment, as the data seems to suggest.

Nevertheless, the data does illustrate some basic points. The action of the machine itself obviously does damage the blood to an extent, as the test free Hb levels are consistently higher than the control ones. At least for the first 10 hours, this damage is within acceptable limits for clinical use. However, the test levels at 11 and 12 hours (111.3 and 115) are unacceptably high. The rubber bung had become partially detached from the syringe plunger by this point in the test, and this could explain the sudden increase in free Haemoglobin. In all types of test the bung detached after approximately 12 hours of use. This was found to be caused by a large increase in friction between the bung and the inside of the syringe barrel. This is thought to be caused by some kind of physical or chemical process

occurring at the interface between the blood and the material of the syringe. Some component of the blood could be absorbed by either the plastic or the rubber, resulting in an increase in friction. Replacement of the syringe always reduces the friction straight away, so the problem is not a result of some sort of degradation in the blood itself. The increase in friction could well be connected to the increased rate of destruction of the red blood cells.

These problems were one of the factors that led to the redesign of the system to use a different type of syringe (see chapter 7). It was decided to repeat the haemolysis test once the redesign had been completed. An identical method was used. The results of this test are shown in table 11.5 and figure 11.12. There are several differences between the two tests that are worth noting. The control sample shows very little increase in free haemoglobin over 12 hours, and the levels themselves are consistently lower than in the first test. The value at 7 hours is much higher than the other values - it can be assumed that this reading is in error. The data for the test sample is more consistent than in the first test. It shows a slower, steady rise in haemoglobin levels over 12 hours. The final values are better, being below 100 mg/dl. The sudden increase seen in the final 2 hours of the first test does not appear here. This would lend weight to the theory that this rise was caused by the separation of the bung from the plunger leading to accelerated blood damage.

A simple calculation from these results shows how significant these levels of blood damage would be in a clinical situation. For example, an 800 g baby would have approximately 60 ml of blood, containing 7.8 g of haemoglobin in total. Assuming a haematocrit of 50 %, there would be 30 ml of plasma in total.

Take the 12 hour figure from the second test : 92.3 mg/dl. So 30 ml of plasma would give:

$$\text{Free Hb} = 92.3 \times \frac{30}{100} = 27.7 \text{ mg}$$

So, after 12 hours the percentage of the total haemoglobin that has been lost can be calculated:

$$\text{percentage loss} = \frac{27.7}{7800} \times 100\ \% = 0.36\ \%$$

This is a very small proportion of the total haemoglobin, and is well within safe limits.

11.4 Blood Access Testing

Further modifications were made to the algorithm for blood withdrawal. It was decided that halving the withdrawal rate every time there was resistance to flow was too much of a reduction in flow rate. The code was changed so that every time the pressure fell below a pre-set limit, the rate was reduced by 1 ml/min, starting from 10 ml/min. The delay before restarting blood withdrawal was kept. A recycling feature was added. Every 15 minutes the blood withdrawal rate is reset back to 10 ml/min, regardless of its current rate. The reason for this is as follows. Temporary occlusion of the line can occur for a variety of reasons. One example is movement of the baby. A small change of position can restrict the flow through the line. But then the baby could move again, opening up the line. If the recycling feature was not included, the withdrawal rate would remain unnecessarily low.

In vivo tests were done to see how the algorithm performed in practice. Two parameters were varied. The trigger pressure was varied between 100 mm Hg and 500 mm Hg. The delay time was varied between 2 seconds and 10 seconds. Blood withdrawal can fail simply because the vein has been 'sucked dry' of blood, allowing the venous walls to collapse. The delay allows time for vascular refilling to occur. It was hoped to find out the minimum time that still allows refilling to occur.

The system was tested using adult volunteers. Superficial veins in the wrist were chosen for blood access, as these approximate to the size of the major veins found in a premature baby. The veins were cannulated using 22 gauge catheters. These were then connected to the machine using luer locking tubing. A special program and user interface was prepared for these experiments. This recorded the venous pressure readings in stripchart fashion, and allowed the behaviour of the system to be monitored as the experiment was proceeding.

A total of four venous access sites were tested, with varying degrees of success. One site occluded too quickly for any testing to be done. The system withdrew blood successfully from the other three sites. Unfortunately the veins used were too large to provide a stringent test of the program. The rate of 10 ml/min was easily achieved, and the flow of blood had to be artificially reduced by applying pressure to the site of cannulation. However, some useful information was obtained and the practicality of the system was established.

11.5 Clinical Use of System

The first clinical use of the system involved a patient suffering from a mitochondrial cystopathy, a genetic metabolic disorder of the mitochondria. Mitochondria are the sites of energy production within the cell. This particular genetic defect disrupts the respiratory pathway and results in excessive production of lactate molecules. High levels of lactate result in a variety of serious medical problems, and if the lactate level is not controlled it is usually a fatal condition.

Although the system was designed for the treatment of acute renal failure, it is equally applicable to metabolic disorders of this type. The principle of operation is the same. In acute renal failure, metabolites that are normally removed by the kidney are removed by dialysis. In the same way, unwanted metabolites that result from metabolic disorders can also be removed.

11.5.1 *Modifications to System*

Some modification of the operating system was necessary. The baby was not particularly premature (8 months) and it was important to reduce the lactate levels as quickly as possible. In view of these factors it was decided to change the parameters of both blood withdrawal and filtration, to increase the clearance of lactate. The maximum rate of blood withdrawal was increased from 10 ml/min to 30 ml/min. The same rate reduction was used, i.e. in steps of 1 ml/min. Blood return was also changed. Before, it was set to always return at 10 ml/min, regardless of the rate of withdrawal. This was changed so that the return rate is always the same as the withdrawal rate. The rate of blood flow through the filter was changed from 10 ml/min to 20 ml/min, and the filtration time was reduced from 4 minutes to 2 minutes. It was thought that since blood access would be good, a reduction in filtration time would result in an overall increase in clearance, because of the increase in effective blood flow rate from the patient.

11.5.2 *Treatment*

The baby was delivered by caesarean section at 30 weeks gestation. As treatment commenced she was suffering from a degree of heart failure, as a result of the metabolic disorder. Two catheters were inserted soon after birth, one into an

umbilical vein and one into an umbilical artery. The access is shown in figure 11.13. The umbilical vein catheter was used for access. The baby weighed 1.7 kg. A safe working blood volume was calculated from this figure. Assuming a total blood volume of 85 ml/kg, there was 144.5 ml of blood in total. If it is safe to remove 10 % of this at any one time, then the upper limit for the working blood volume is 14.5 ml. It was decided to start at 5 ml and then increase to 10 ml, allowing a wide margin for error. The ultrafiltration rate was set to zero. No ultrafiltration was necessary as the patient's kidneys were assumed to be functioning normally. The dialysate flow rate was set to 400 ml/h. With some difficulty the circuit was primed and treatment began. Due to an error in the heparinisation calculations, the circuit clotted very quickly. This was detected successfully by the operating system - the alarm went off and the machine stopped. The circuit was replaced, and treatment recommenced. The initial working blood volume of 5 ml caused no adverse effects, so it was increased to 10 ml. This value was also well tolerated. Figures 11.14 and 11.15 show the system in operation.

Treatment continued for 8 hours without any problems occurring. At this point the umbilical venous catheter became partially occluded, due to a change in its position within the vein. Several attempts were made to open it up again. The pressure display proved very useful during this procedure, as it gave a continuous readout of the pressure in the line while it was being adjusted. It was decided that the venous line had become too unreliable, so the access was switched over to the umbilical arterial catheter. The system ran for a further 12 hours without incident.

After 20 hours of dialysis, the patient's lactate levels had been successfully reduced. However, the heart failure that she had been born with proved to be untreatable, and the patient died.

11.5.3 *Results and Conclusions*

The new effective blood flow rate obtained from the system modifications can be calculated and compared with the old value:

Assume a working blood volume of 10 ml
and a 20 ml / min withdrawal rate.
For the old system:
withdrawal rate = 10 ml / min
$\Rightarrow t_w = 10/10 = 1$ min.
return rate = 10 ml / min
$\Rightarrow t_r = 1$ min.
filtration time, $t_f = 4$ min.
Reset time is zero, so
effective blood flow rate, Q_B, is given by

$$Q_B = \frac{\text{blood volume}}{t_w + t_r + t_f} = \frac{10}{1+1+4} = 1.66 \text{ ml / min.} \qquad (11.10)$$

For the new system:

withdrawal rate = 20 ml / min
$\Rightarrow t_w = 10/20 = 0.5$ min.
return rate = 20 ml / min
$\Rightarrow t_r = 0.5$ min.
filtration time, $t_f = 2$ min.
Reset time is zero, so
effective blood flow rate, Q_B, is given by

$$Q_B = \frac{\text{blood volume}}{t_w + t_r + t_f} = \frac{10}{0.5+0.5+2} = 3.33 \text{ ml / min.} \qquad (11.11)$$

So, the blood flow rate has been doubled. Lactate measurements were taken during dialysis so that clearances could be calculated:

Plasma concentrations:
lactate concentration pre-filter, C_{pre} = 6.8 mmol/l
lactate concentration post-filter, C_{post} = 4.7 mmol/l
lactate concentration in dialysate, C_{dial} = 3.2 mmol/l

$$\text{Clearance, } D = Q_B \times \frac{(C_{pre} - C_{dial}) - (C_{post} - C_{dial})}{(C_{pre} - C_{dial})} \qquad (11.12)$$

$$= 3.333 \times \frac{(6.8 - 3.2) - (4.7 - 3.2)}{(6.8 - 3.2)}$$

$$= 1.94 \text{ ml/min}$$

This calculation is based on concentration values that are over and above the concentration of lactate contained in the dialysis fluid. This value of clearance is almost double the best that was achieved before the modifications were made.

With this amount of clearance, the patient's lactate levels were very quickly brought under control. The time taken to reduce the lactate level to half of its initial value can be calculated:

$$M_T = V_B \times C_{plas} \qquad (11.13)$$

where
M_T = total lactate mass in bloodstream,
V_B = Total blood volume = 144.5 ml,
and C_{plas} = Plasma lactate concentration
The rate of change of total mass is equal to the rate of removal by dialysis (assuming no lactate is generated). So, by the definition of clearance:

$$\frac{dM_T}{dt} = -D \times C_{plas} \qquad (11.14)$$

Substituting from (11.12),

$$\frac{d(V_B \times C_{plas})}{dt} = -D \times C_{plas} \qquad (11.15)$$

$$\Rightarrow \frac{dC_{plas}}{dt} = -\frac{D}{V_B} \times C_{plas} \qquad (11.16)$$

$$\Rightarrow \int_0^t dC_{plas} = -\frac{D}{V_B} \int_0^t C_{plas} dt \qquad (11.17)$$

Integration by standard methods gives:

$$C_t = C_0 e^{-\frac{D}{V_B}t} \qquad (11.18)$$

where C_0 = initial plasma lactate concentration

C_t = plasma lactate concentration at time t

Substituting $\frac{C_t}{C_0} = 0.5$ into this equation gives

$$t_{0.5} = \ln 2 \times \frac{V_B}{D} = 0.693 \times \frac{144.5}{1.94} = 52 \text{ min} \qquad (11.19)$$

Although it is difficult to draw definite conclusions from just one treatment, these results compare very favourably with other published results. For instance, Schaefer[93] reports a 50 % reduction time of 7.1 hours (± 4.1 hours) in neonates being treated by continuous venovenous haemodialysis.

The blood access algorithm proved to be very successful in practice. The access rate settled to around 22 ml/min for both the venous and the arterial catheters. Observation of the pressure trace during the delay times while the withdrawal rate was being reduced revealed that the delay of 5 seconds was unnecessarily large. Vascular refilling was taking place well within this time. The delay time was therefore reduced to 2.5 seconds.

Other refinements were added as a result of the experience gained in the first use of the machine. Circuit priming was made an integral part of the initialisation procedure. A simple manual control interface was written to allow the user to turn both 3 way tap drivers to any desired combination of positions. The main use of this would be to allow the blood remaining in the circuit to be flushed back into the patient at the end of a treatment session, without having to take the circuit off the machine first.

Summary

The testing of the system functioning as a haemodialyser established the clearances that could be obtained in this mode of operation. They were found to meet and exceed the clinical requirements for the system. Haemolysis testing established the amount of damage being caused to the patient's blood. This was found to be within acceptable limits. The blood access algorithm was tested using the small superficial wrist veins of adult volunteers. Finally, the system was used successfully to dialyse a baby suffering from an in born genetic metabolic disorder.

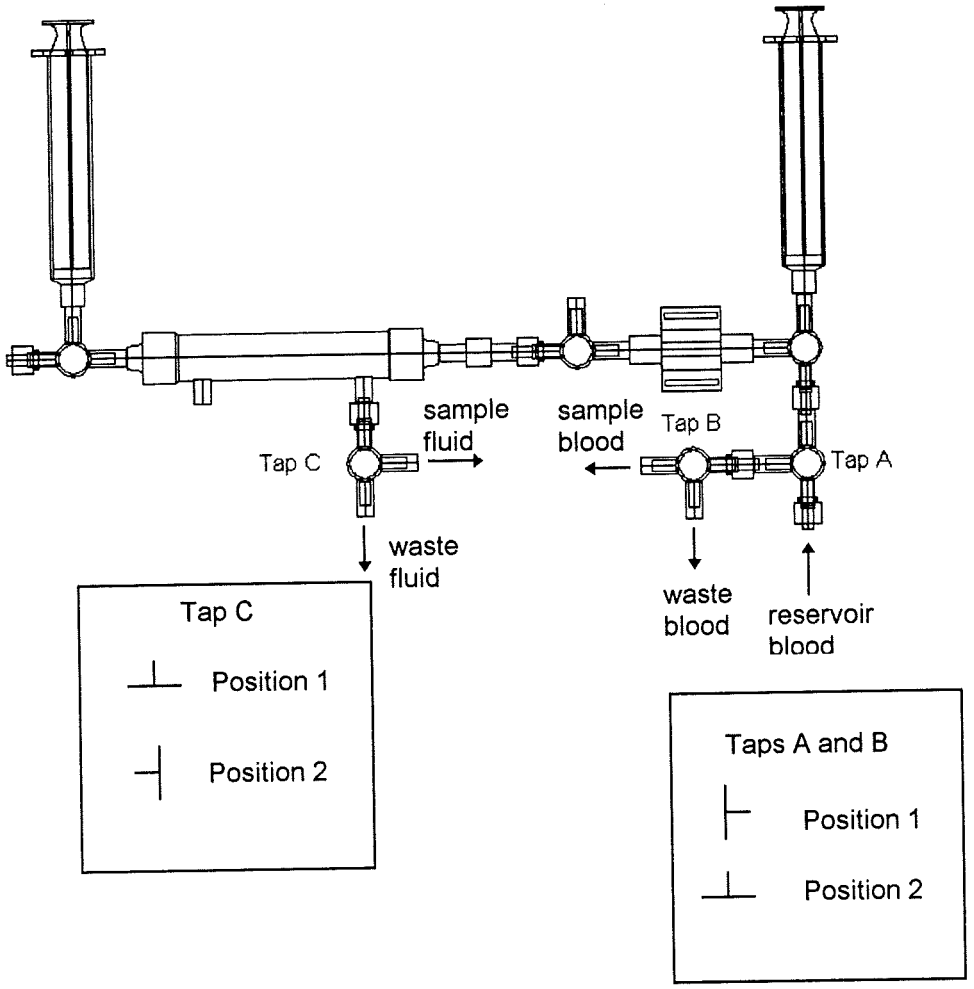


Figure 11.1 Circuit for Dialysis Testing

test no.	working blood volume ml	dialysate flow rate ml/h	cycle time min	plasma K+ mmol/l	plasma urea mmol/l	plasma creat. µmol/l	fluid K+ mmol/l	fluid urea mmol/l	fluid creat. µmol/l	clearance plasma K+	clearance plasma urea	clearance plasma creat.	clearance fluid K+	clearance fluid urea	clearance fluid creat.	mean clearance
1	3	100	2.23	2.4	14.1	180	0.8	5.3	49	0.278	0.273	0.146	0.307	0.400	0.366	0.295
2	3	100	3.17	2	11.7	153	0.7	4.4	39	0.251	0.250	0.167	0.263	0.332	0.291	0.259
3	3	100	5.3	1.5	8.2	118	0.6	3.8	33	0.192	0.200	0.150	0.219	0.287	0.247	0.216
4	3	30	5.3	1.7	10.2	137	1	7.4	73	0.175	0.171	0.122	0.118	0.167	0.164	0.153
5	3	60	5.3	1.6	9.6	123	0.8	5.6	51	0.184	0.180	0.142	0.184	0.253	0.229	0.195
6	5	50	5.55	1.7	10.2	136	1.2	8.3	82	0.279	0.272	0.197	0.241	0.313	0.306	0.268
7	5	100	5.55	1.6	8.9	125	0.8	5.6	51	0.293	0.302	0.222	0.307	0.422	0.381	0.321
8	5	600	5.55	1.2	6.2	107	0.2	1.2	9	0.346	0.364	0.263	0.263	0.543	0.404	0.364
9	7	70	6.18	1.7	10.6	126	1.1	8.1	77	0.351	0.331	0.276	0.307	0.428	0.403	0.349
10	7	140	6.18	1.4	7.9	112	0.8	5.6	47	0.401	0.408	0.316	0.430	0.591	0.492	0.440
11	10	100	6.63	1.8	11	128	1.1	7.7	70	0.445	0.425	0.360	0.439	0.581	0.523	0.462
12	10	200	6.63	1.5	8.7	111	0.7	4.9	40	0.512	0.513	0.425	0.526	0.739	0.598	0.552

Table 11.2 Dialysis Test Results for 10 ml Syringe

Working blood volume ml	dialysate flow rate ml/h	cycle time min	Blood Flow Rate (ml/min)	plasma K+ mmol/l	plasma urea mmol/l	plasma creat. µmol/l	fluid K+ mmol/l	fluid urea mmol/l	fluid creat µmol/l	clearance plasma K+	clearance plasma urea	clearance plasma creat	clearance fluid K+	clearance fluid urea	clearance fluid creat	means
3	30	5.3	0.57	10.6	10.3	153	8.8	9.4	140	0.156	0.130	0.143	0.236	0.305	0.292	0.210
5	50	5.55	0.90	8.2	8.1	125	6	6.7	92	0.334	0.297	0.300	0.260	0.363	0.319	0.312
7	70	6.18	1.13	8.4	8.2	134	6.2	6.7	91	0.411	0.368	0.348	0.378	0.508	0.442	0.409
10	100	6.63	1.51	8.7	8.3	131	5.5	6.2	81	0.530	0.483	0.476	0.473	0.671	0.563	0.533
15	150	8	1.88	8.4	7.8	129	4.7	5.4	67	0.681	0.643	0.603	0.597	0.877	0.698	0.683
20	200	8	2.50	8.5	8	129	4.6	5	64	0.898	0.835	0.804	0.777	1.082	0.889	0.881
3	60	5.3	0.57	7.7	7.6	121	4.8	4.9	67	0.221	0.199	0.195	0.244	0.318	0.279	0.243
5	100	5.55	0.90	7.1	6.8	118	3.3	3.5	45	0.374	0.350	0.318	0.265	0.379	0.313	0.333
7	140	6.18	1.13	8.5	8.4	135	3.8	4.3	50	0.407	0.358	0.344	0.438	0.652	0.486	0.447
10	200	6.63	1.51	6.8	6.4	109	3.6	4.1	55	0.643	0.613	0.572	0.587	0.887	0.764	0.678
15	300	8	1.88	6.4	5.8	105	2.9	3.4	40	0.829	0.812	0.733	0.682	1.104	0.833	0.832
20	400	8	2.50	6.1	6.2	109	3	3.2	40	1.135	1.038	0.948	0.947	1.385	1.111	1.094

Table 11.3 Dialysis Test Results for 25 ml Syringe

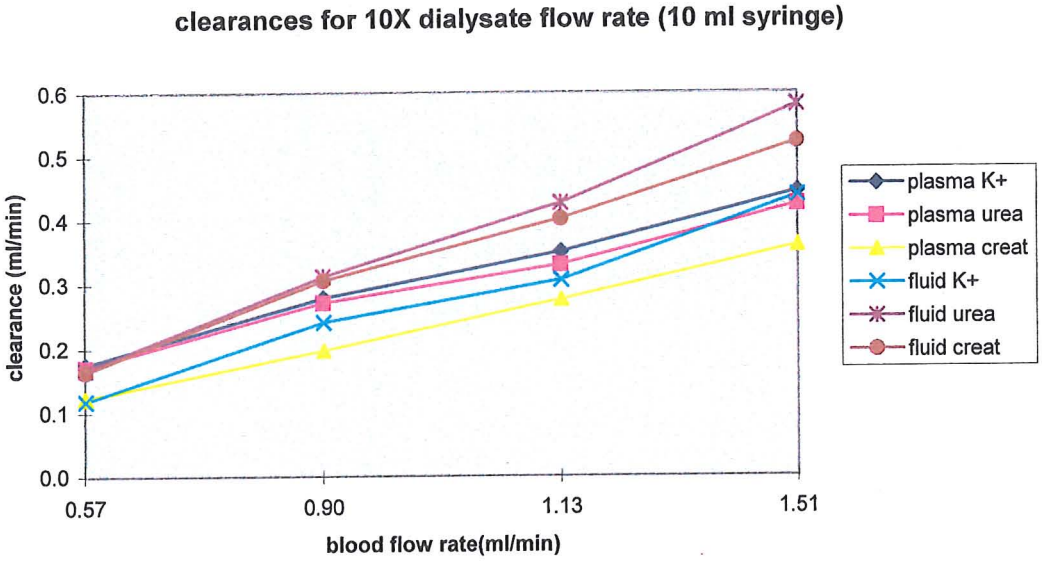


Figure 11.2

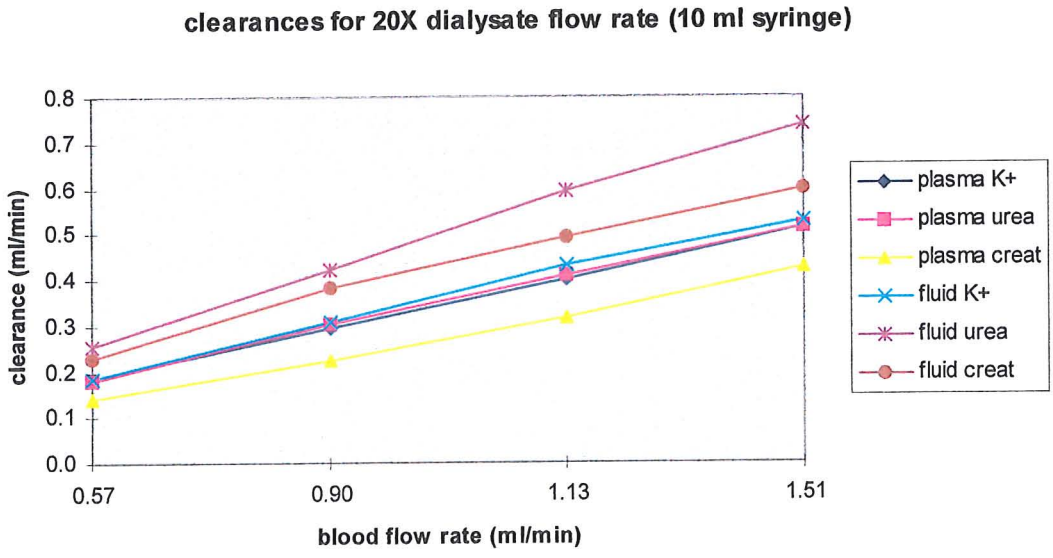


Figure 11.3

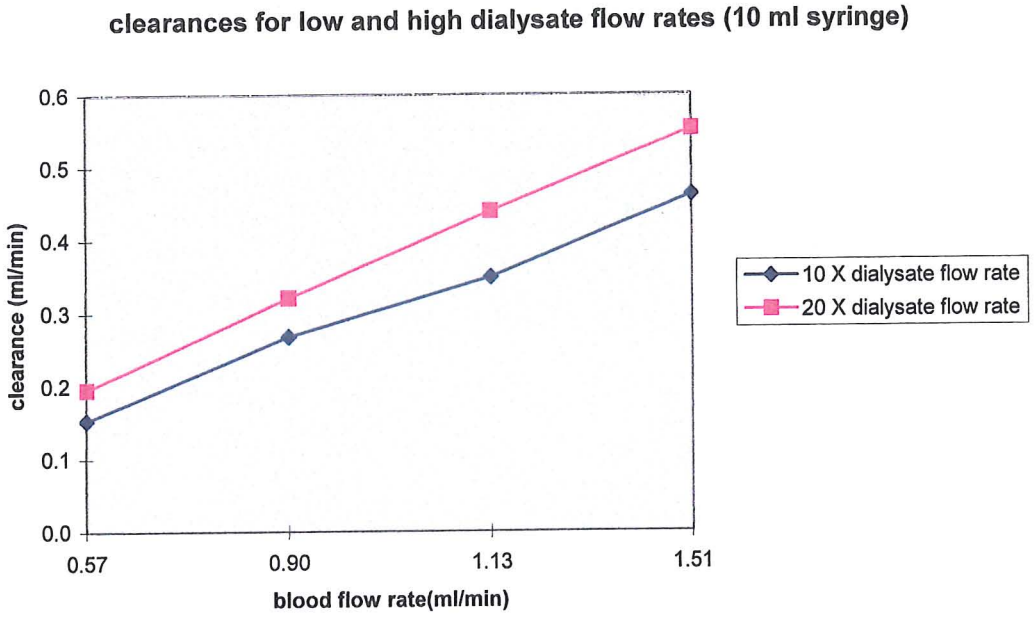


Figure 11.4

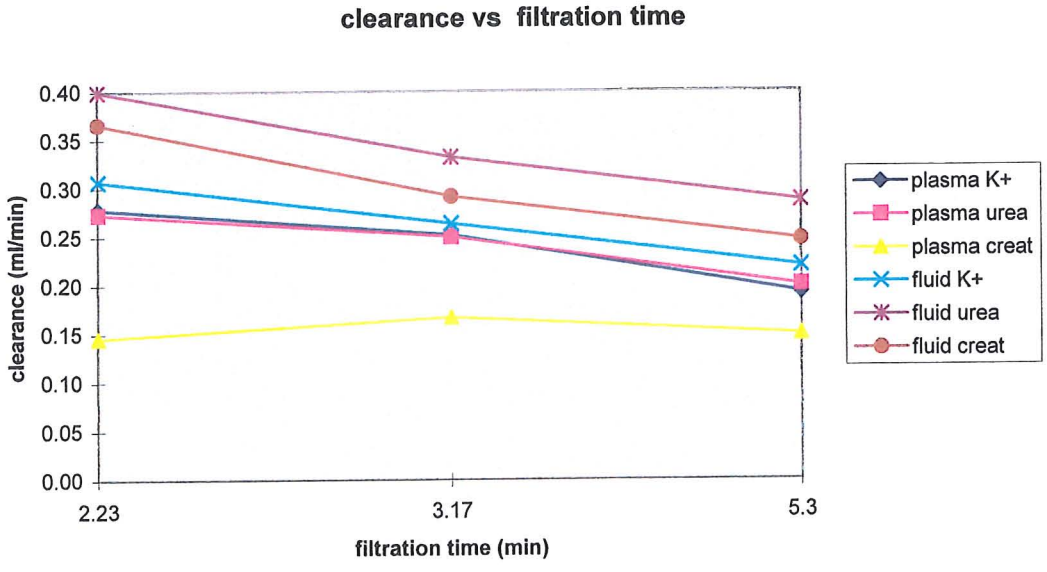


Figure 11.5

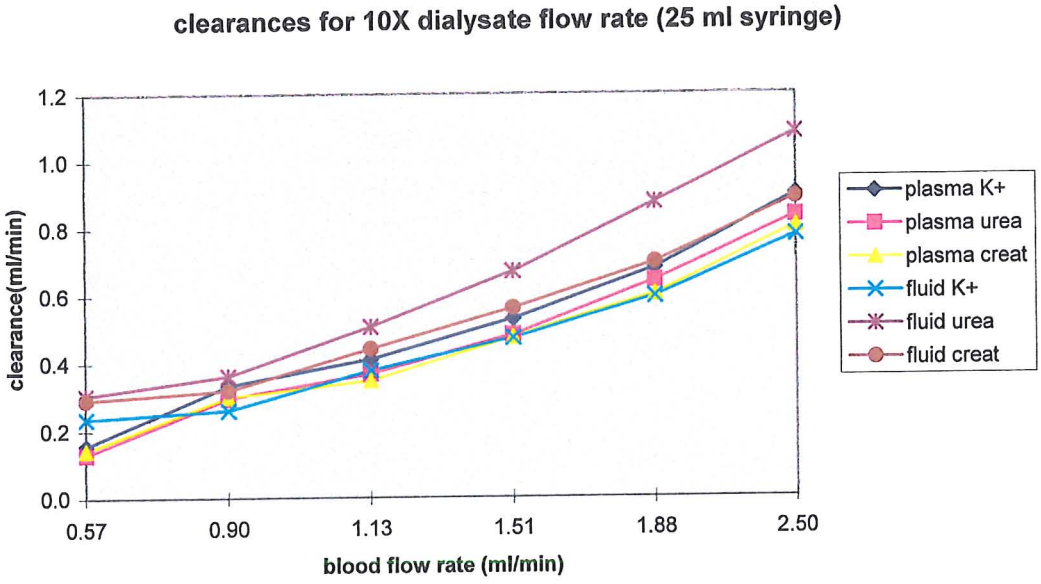


Figure 11.6

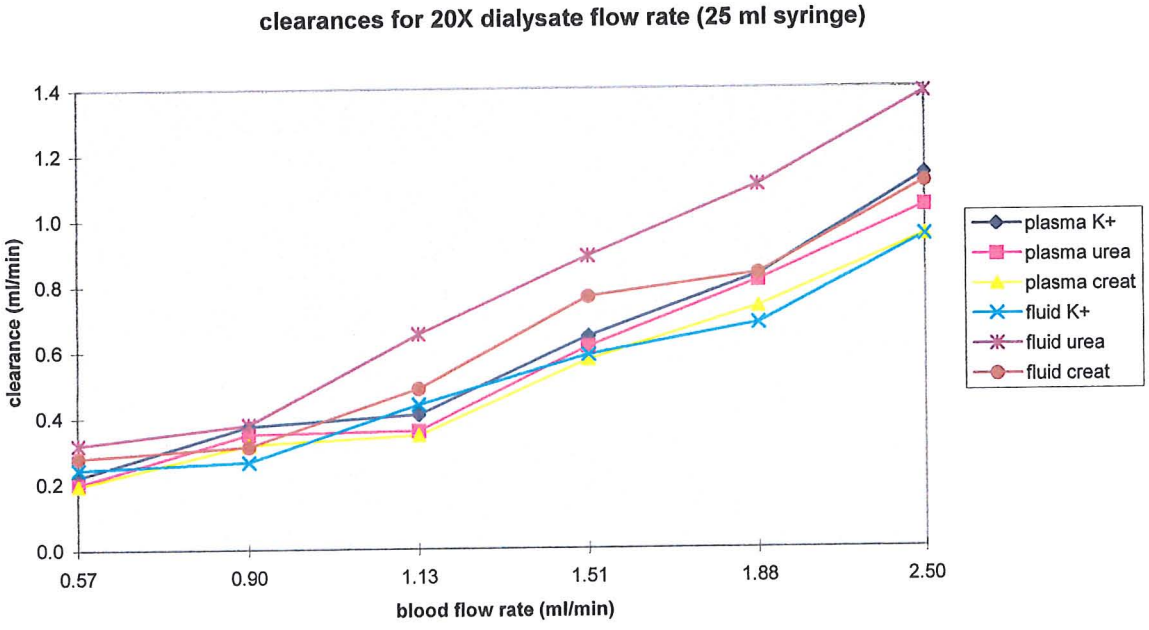


Figure 11.7

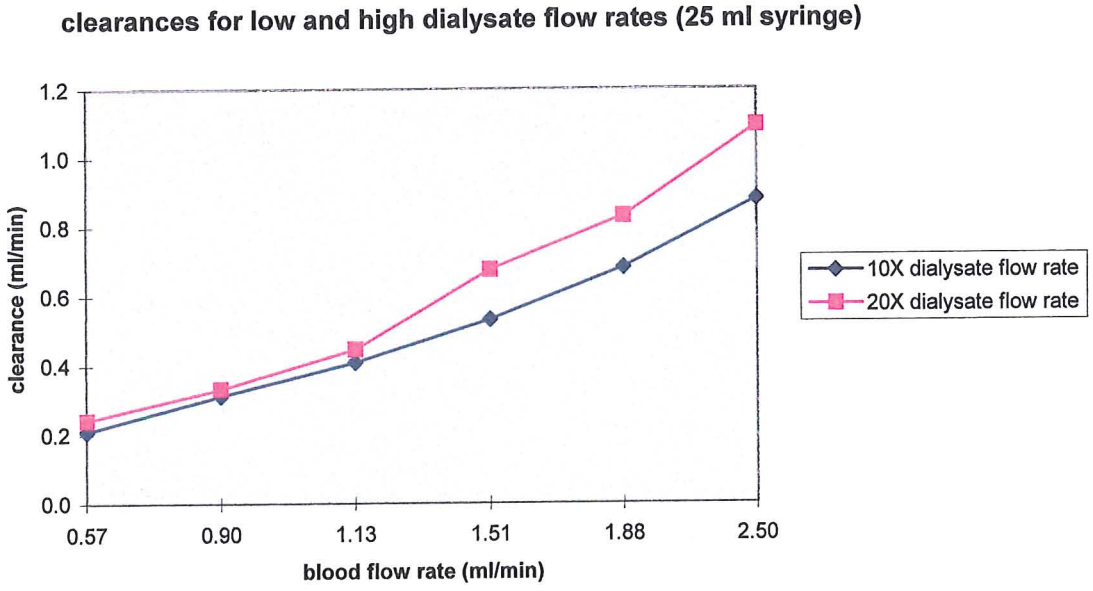


Figure 11.8

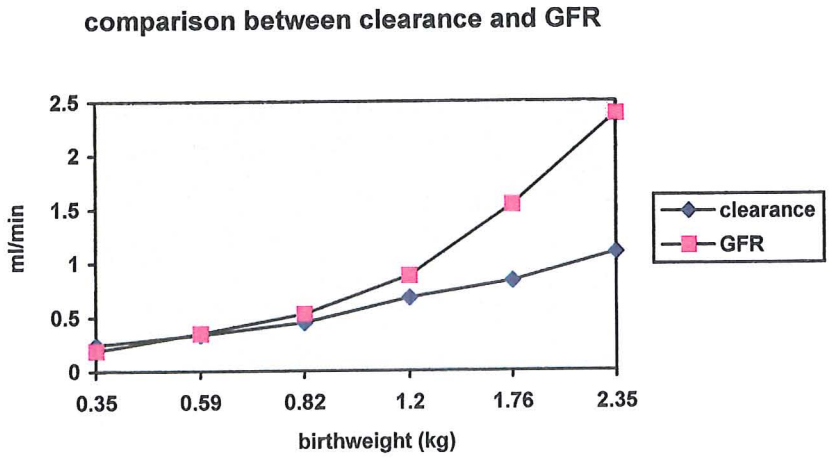


Figure 11.9

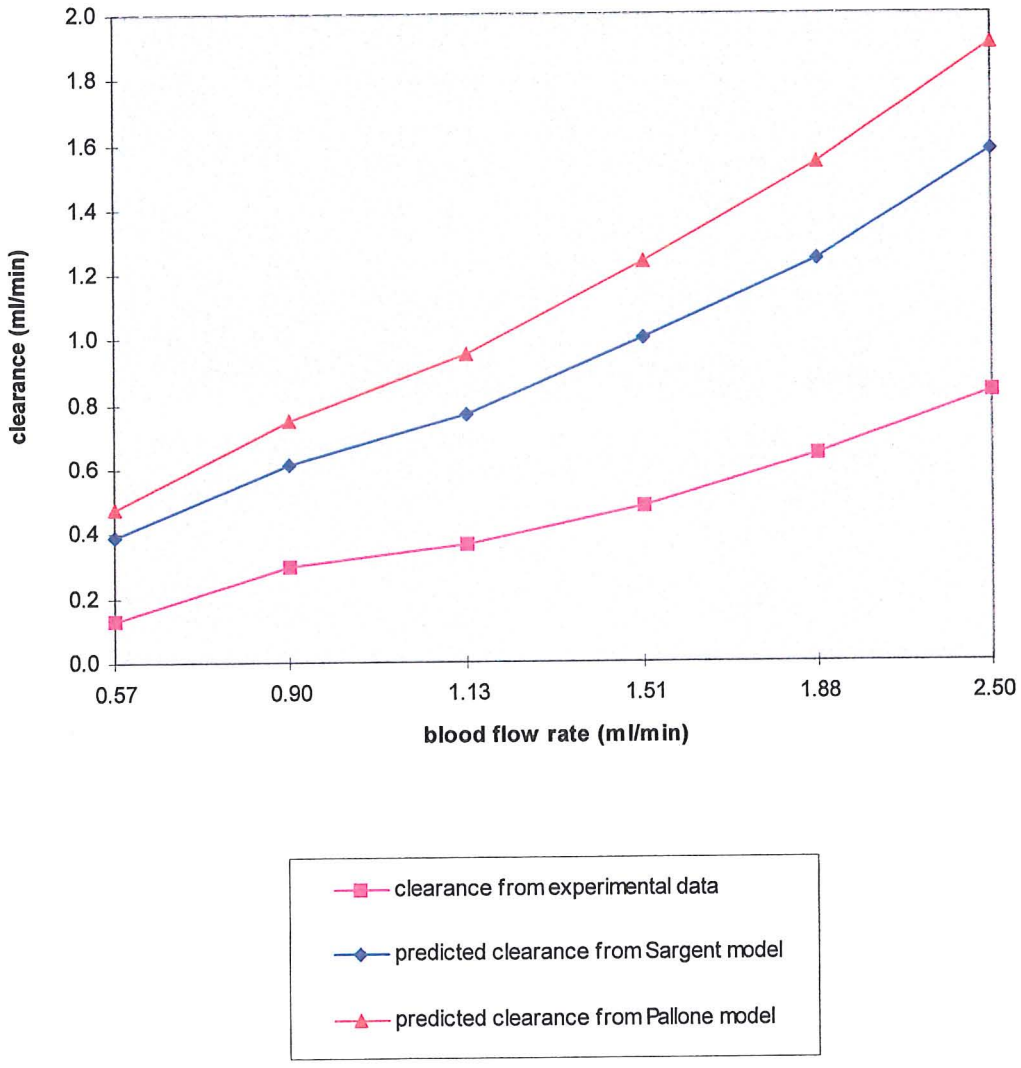


Figure 11.10 Comparison of Experimental Data with Models of Clearance

time in hours	0	1	2	3	4	5	6	7	8	9	10	11	12
control free Hb mg/dl	17.3	13.5	19	22	21.5	21.8	27	27.9	24.9	25.7	42.9	33.3	35.1
test free Hb mg/dl	17.3	16.3	16.2	17.3	29.2	44	60.6	36.4	71.3	65.3	51.8	111.3	115

Table 11.4 Haemolysis Test Results for 10 ml syringe

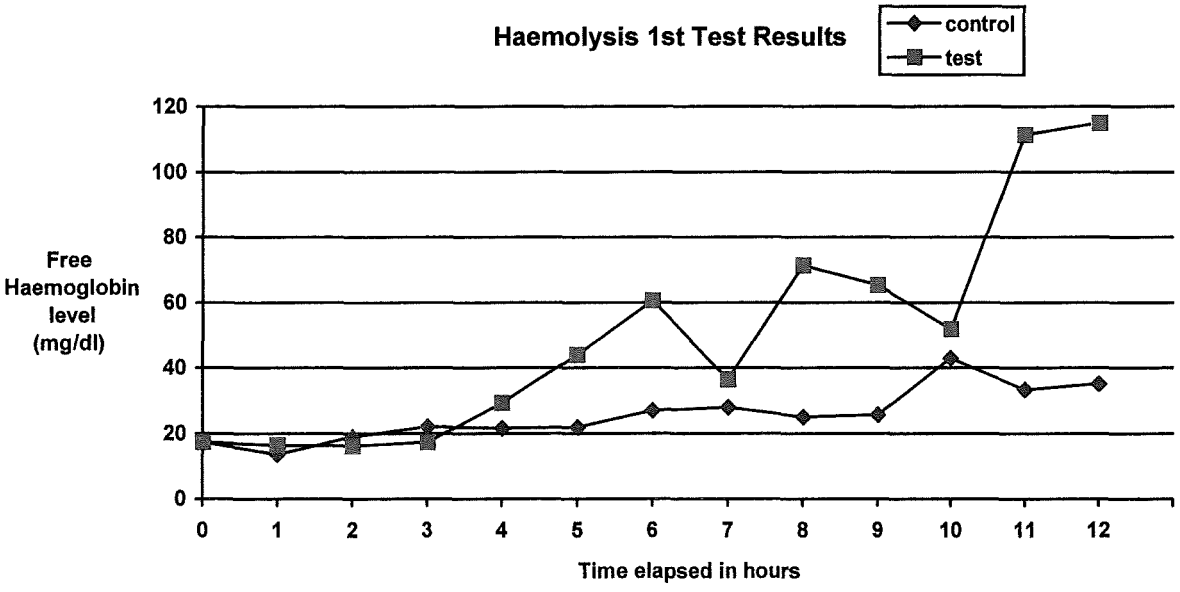


Figure 11.11

time in hours	0	1	2	3	4	5	6	7	8	9	10	11	12
control free Hb mg/dl	4.3	3.3	2.3	5.6	7.7	4.4	5	24.1	5.7	5.4	5.9	6	14.2
test free Hb mg/dl	4.3	22.3	41.9	45.4	47.4	55.4	57.4	64.3	66.3	70.3	77.8	83.1	92.3

Table 11.5 Haemolysis Test Results for 25 ml Syringe

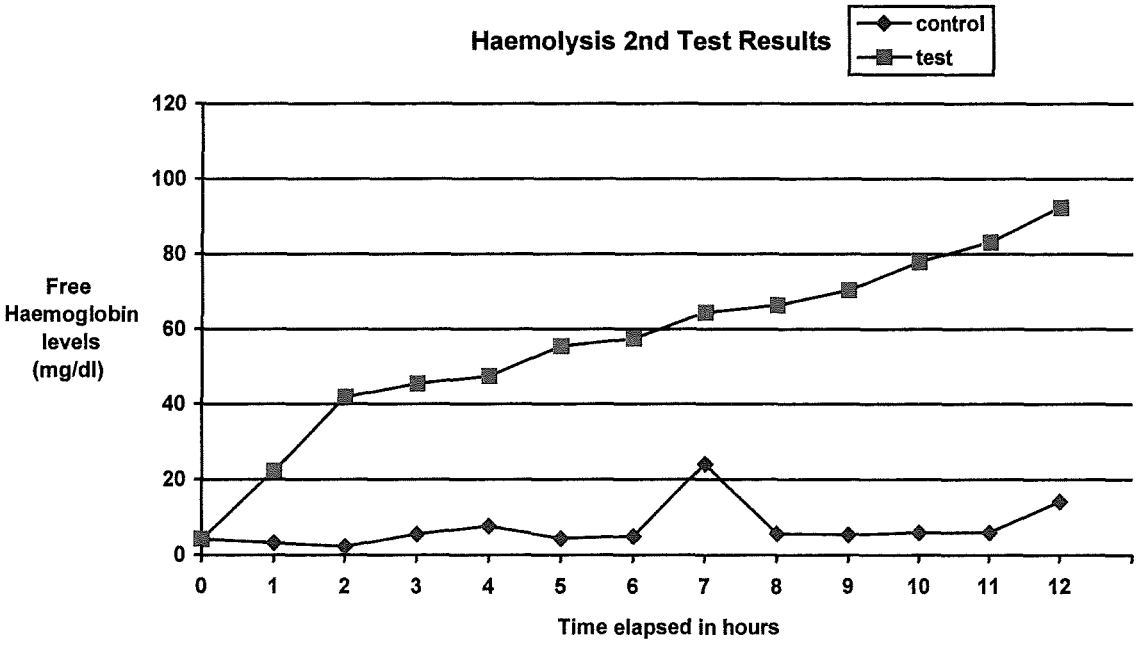


Figure 11.12

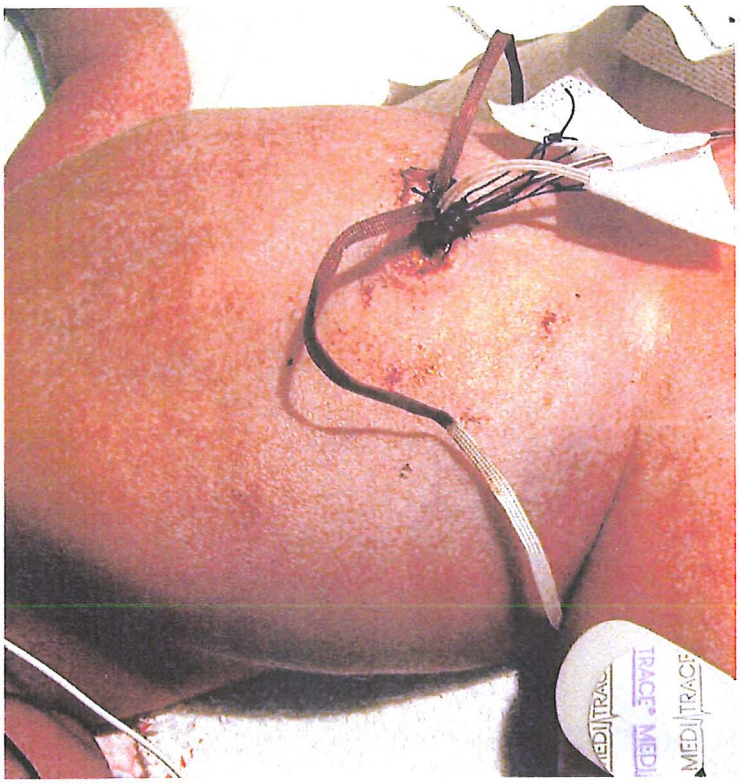


Figure 11.13 Umbilical Access

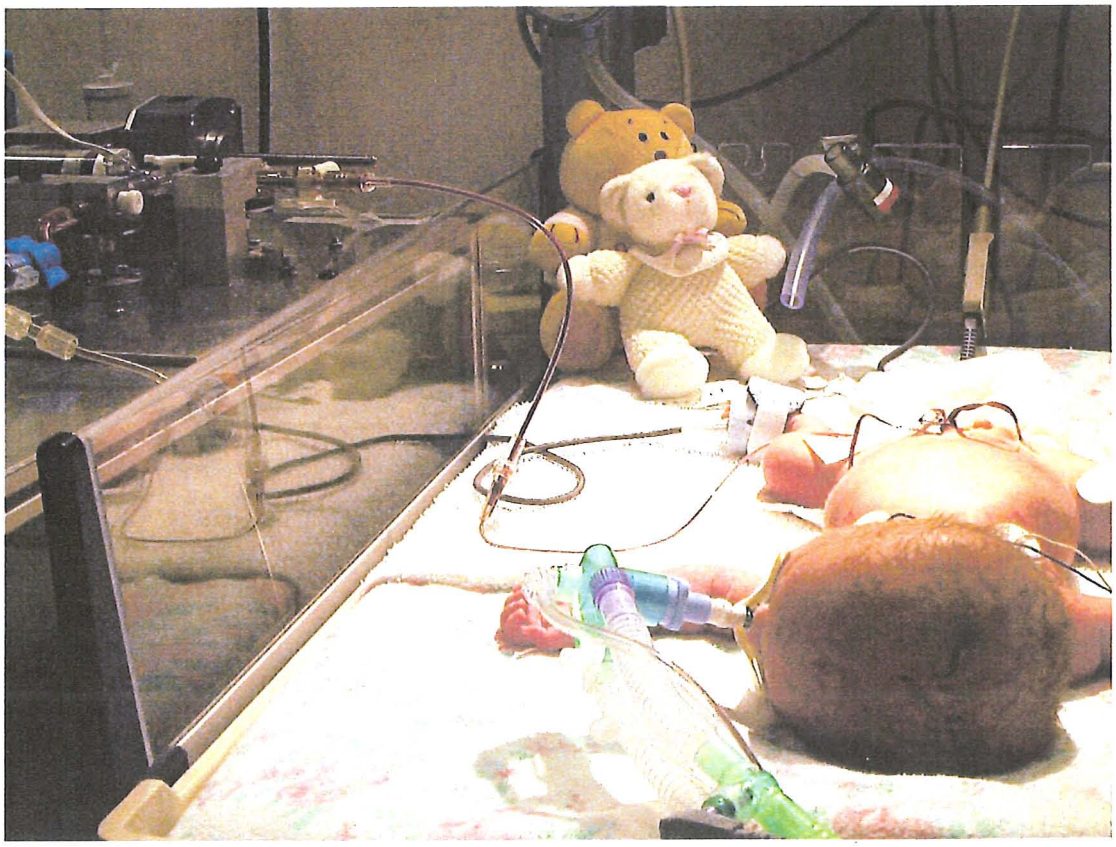


Figure 11.14 Close Up of System in Operation

Figure 11.15 Overview of System in Operation

CHAPTER 12. CONCLUSIONS AND PROPOSALS FOR FURTHER WORK

Introduction

This final chapter presents the conclusions that have been reached as a result of the work done, and a discussion of these. Proposals for future work are also given.

12.1 Conclusions

- It has been shown that it is feasible to automate syringe driven haemodialysis. A novel prototype system has been developed, extensively tested and proven to perform to the specification originally drawn up for the project.

- To date it has been used successfully to treat one patient suffering from a genetic metabolic disorder.

- The original aims of the project have been expanded. Initially it was intended only for use in treating premature babies suffering from acute renal failure. The system has performed better than expected, so the scope of its potential application has increased. This now includes heavier babies (up to 5 kg), and babies suffering from a variety of metabolic disorders.

- A comprehensive literature review was undertaken. This failed to find any work of a similar nature that had been undertaken before. The current project represents a unique contribution to the subject of dialysis in neonates.

- The mechanical, electromechanical and electronic components of the system were designed, constructed and tested successfully. These were then integrated into the system as a whole.

- A comprehensive control and safety system was developed using C and Labwindows/CVI software. The safety system was developed using a single fault analysis of the entire machine.

- The testing of the system concerned three major aspects of its operation. Control of ultrafiltration rate was assessed to ascertain that it fell within the ± 5% accuracy demanded by the specification. The clearance rates that could be achieved were established with *in vitro* testing, and found to be more than adequate for clinical purposes. The haemolysis produced by the system was tested and found to be well within safe limits.

- The system directly controls ultrafiltration rate. This represents a significant advantage over most conventional circuits, which can only indirectly control ultrafiltration rate, with a consequent loss of accuracy.

- The ultrafiltration rates that can be obtained are significantly higher than those that can be achieved with conventional circuits.

- The clearances that can be achieved are very similar to those that can be obtained with conventional circuits.

- The total cost of the prototype system was approximately £5,400. This was well within the available budget of £15,000.

- The system that has been produced is a prototype. A second, more clinically acceptable version is planned. This would include a variety of improvements that are detailed in section 12.2.

- In summary, a system has been produced that is superior to any other method of treatment that is currently available for babies suffering from acute renal failure and metabolic disorders.

The specification was met or exceeded in all but one area. Some work was done to attempt to accommodate different sizes of haemofilter in the extracorporeal circuit. This would still be possible with some modification of the existing system. It would be necessary to redesign the top cover of the casing so that the system was splash proof when haemofilters of a smaller length were fitted.

The clinical data obtained from the patient that was treated for mitochondrial cystopathy confirmed the performance figures that were obtained from the *in vitro*

studies. Lactate clearances of 1.94 ml/min were achieved. The lactate levels in the blood were reduced by 50 % in approximately 1 hour. This is a significant improvement on other published data [93].

The clearance studies showed that the performance of the system was sufficient to be able to dialyse babies up to a weight of 5 kg. Using the same empirical relationships [91] as in chapter 11, the glomerular filtration rate of a 5 kg baby would be approximately 8.25 ml/min. A baby of this size would have a total blood volume of 425 ml, so 42.5 ml could be safely withdrawn for dialysis. The maximum that can be withdrawn by the system is 25 ml, and this could therefore be safely used with a baby of this size. Extrapolation from the clearance studies in chapter 11 shows that a working blood volume of 25 ml could be expected to produce a clearance of roughly 1.3 ml/min - this is 16 % of the GFR. Therefore the system can produce sufficient clearance to dialysis a baby of this size.

The literature search that was undertaken failed to find any evidence of similar work having been undertaken before. Two systems have been described [78, 79] which bear only a very superficial resemblance to the syringe driven dialysis system described here.

Direct control of ultrafiltration rate is a significant advance over existing methods of dialysis for premature babies. In CAVH and CVVH, it can be very difficult to obtain sufficient ultrafiltration, because of low systemic blood pressure and difficulties of vascular access. It can also be very difficult to control the ultrafiltration rate. The syringe driven system completely circumvents these problems. The ultrafiltration rate that is needed by the clinician can be simply entered into the user interface, and obtained to an accuracy of ± 5%. The maximum ultrafiltration rate is limited by two different factors from those above. The first is the amount of blood that can be safely withdrawn from the baby. The second is the haematocrit of the blood in the extracorporeal circuit. If too much filtrate is withdrawn from a sample of blood, the haematocrit can become high enough to cause a significant risk of clotting in the extracorporeal circuit. However, these factors are less limiting than the factors at work in CAVH and CVVH. As was shown in chapter 10, the syringe driven system can achieve approximately 3 times the ultrafiltration rate of conventional circuits.

The clearance data showed a similar performance to conventional circuits. This was expected. The alternating countercurrent and concurrent flow of blood that occurs during the dialysis phase of the operating cycle could potentially reduce

clearance by reducing the mean concentration gradient across the haemofilter membrane. However, this effect is eliminated by using a high dialysate flow rate, which maintains a favourable concentration gradient. The flow rate needed is still viable from a clinical point of view. Under typical treatment conditions, a 5 litre bag of dialysate solution can be expected to last 12 hours.

12.2 Proposals for Further Work

Second Prototype System

A second prototype machine is currently being developed by the Medical Physics department of the Royal Victoria Infirmary. It will implement several improvements on the existing design. Closer adherence to IEC 60601, the safety requirements for medical electrical equipment is needed, with particular attention to part 2 [90], the requirements for haemodialysis equipment. The improvements that are proposed are detailed below. The first few are relevant to safety, the later ones address other aspects of the system design.

It is very difficult to guarantee the safety of computer systems and their software. This problem can be avoided by providing hardware interlocks on the machine that prevent any malfunction in the computer system resulting in a dangerous condition. Any fault arising would result in the machine being set to a fail safe condition independently of the computer control system.

The design of the casing and overall layout of the machine needs to be improved. As far as possible, all components should be inside the casing (e.g. the stepper motors), so that the extracorporeal circuit attaches to a surface with a minimum of protruding components. This makes the machine easier to clean and disinfect. The orientation of the machine should be changed so that the circuit attaches to a vertical surface instead of a horizontal one. This would make splash proofing of the casing a lot easier to achieve. The overall size of the machine could be reduced by changing the orientation of the syringe drivers, so that they lie along the haemofilter (one above and one below) instead of at right angles to it. This would provide a significant size reduction.

It was decided not to include an air bubble detection system in the prototype device. Commercially available air bubble detection devices are available, but the smallest of these can only be used on 3 mm diameter lines. It was not feasible to

modify them to be used on the 1 mm lines employed by the syringe driven haemodialysis system. However, the second prototype machine should have such a safeguard. This would monitor the venous line, and would shut down the whole system if an air embolism was detected. A device would have to be custom built for the system.

An isolation amplifier should be added between the pressure transducer and its amplifier. Although the transducer is already an isolated component, the addition of a further stage of isolation is customary in medical equipment. A more accurate amplifier that is less susceptible to electrical noise would be a worthwhile improvement.

Although the Hamilton syringes are gas tight, they do not perform well when subjected to negative pressure. At about 400 mm Hg, air starts to leak by the seal. Although this is acceptable in the prototype system, a higher negative pressure limit would improve safety.

The syringes currently used have several drawbacks. Because they are not disposable, they have to be specially sterilised before each use. This is expensive and time consuming. The syringes themselves are costly, at £ 85 each. A cheaper, preferably disposable syringe that can be used for periods of time as long as one week needs to be found.

The stepper motor microswitches should be changed to a solid state photodetection system, similar to that used with the three way tap drivers. A solid state system would be more reliable than one that uses mechanical switches.

Improvement to the volumetric control of ultrafiltration should be considered. While the current system is accurate to within ±5 %, more accuracy would have clinical benefits. Very accurate volumetric sensors are available. It should be possible to mount two of these on the dialysate inflow and outflow lines, and design a closed loop system to control the rate of ultrafiltration very closely.

Replacing the current minitower PC with a laptop would result in a significant size reduction in the whole system. Data acquisition cards that can be fitted to the expansion buses of laptop computers are readily available. This, together with the reduction in size of the machine itself would bring practical clinical benefits. Space is limited in a special care baby unit, and the cot is already surrounded by a large array of bulky equipment. So, the smaller the system is, the easier it is to set up and use.

Further *in Vitro* Studies and System Modelling

The syringe driven haemodialysis system is more difficult to model than conventional haemodialysis circuits because of the unsteady nature of the flow of blood within the system. The development of an unsteady state model could provide very useful information about the system, leading to optimisation of its performance. This would require further *in vitro* studies using the prototype system. The extracorporeal circuit would have to be fitted with further pressure measuring instrumentation to provide more detailed information about its behaviour. A sampling system would have to be developed that would allow local sampling of blood and dialysate at various points in the circuit, so that the model could be compared with experimental data.

Further Clinical Studies

The prototype system is currently only available for use within the Newcastle upon Tyne NHS hospitals trust. With this restriction, a suitable patient will only become available on average once every 6 months. As more patients are treated, more clinical data will become available, leading to a more accurate assessment of the system's performance. The second prototype system will be available to other trusts outside the Newcastle area. This will result in the system being used much more frequently, so patient data will be gathered at a much greater rate.

Commercialisation of System

A patent application protecting the system is currently being planned. If this is successful commercialisation of the system should follow. A thorough study of commercial viability is needed. At present there are 10 regional centres in the UK that provide treatment for renal failure in neonates. The provision of 10 systems is unlikely to be commercially viable, so the system would have to be marketed world wide.

Summary

This final chapter has described the conclusions that have been drawn from the research that has been undertaken, emphasising the successful completion of the development of the prototype syringe driven haemodialysis system. Suggestions for further work have been given. These include details of the second prototype machine, further *in vitro* and *in vivo* studies, system modelling, and the possible commercialisation of the system.

13. References

1. Norman, M.E., and F.K. Asadi, 1979. A prospective study of acute renal failure in the newborn infant. *Pediatrics.* 63: 475 - 9.

2. Stapleton, F.B., 1987. Acute renal failure in neonates: incidence, etiology and outcome. *Pediatric Nephrology.* 1: 314 - 320.

3. Coulthard, M.G., and B. Vernon, 1995. Managing acute renal failure in very low birthweight infants. *Archives of Disease in Childhood.* 73: 187 - 192.

4. Bunchman, T.E., and R.A. Donckerwolcke, 1994. Continuous arterial-venous diahemofiltration and continuous veno-venous diahemofiltration in infants and children. *Pediatric Nephrology.* 8: 96 - 102.

5. Ganong, W.F., 1981. *Review of Medical Physiology.* 10th ed. Lange Medical Publications.

6. Coulthard, M.G., and J. Sharp, 1995. Haemodialysis and ultrafiltration in babies weighing under 1000g. *Archives of Disease in Childhood.* 73: 162 - 165.

7. Macleod, J., C. Edwards and I. Bouchier, eds., 1987. *Davidson's Principles and Practice of Medicine.* 15th ed. Churchill Livingstone.

8. Gottschalk, C.W., and S.K. Fellner, 1997. History of the science of dialysis. *American Journal of Nephrology.* 17: 289 - 298.

9. Van Stone, J.C., 1997. Dialysis equipment and dialysate, past, present and the future. *Seminars in Nephrology.* 17:3 214 - 217.

10. Abel, J.J., L.G. Rowntree and B.B.Turner, 1913. On the removal of diffusible substances from the circulating blood by means of dialysis. *Transactions of the Association of American Physicians.* 28: 51 - 54.

11. Fagette, P., 1999. Hemodialysis 1912 - 1945: no medical technology before its time, part I. *ASAIO Journal.* 79: 238 - 249.

12. Fagette, P., 1999. Hemodialysis 1912 - 1945: no medical technology before its time, part II. *ASAIO Journal.* 45: 379 - 391.

13. Teschan, P.E., 1993. Building an acute dialysis machine in Korea. *ASAIO Journal.* 39:4 957 - 961.

14. Peitzman, S.J., 1997. Origins and early reception of clinical dialysis. *American Journal of Nephrology.* 17: 299 - 303.

15. McBride, P., 1989. Industry's contribution to the development of renal care. *ANNA Journal.* 16: 217 - 226.

16. Mateer, F.M., L. Greenman, and T.S. Danowski, 1955. Hemodialysis of the uremic child. *American Journal of Diseases of Children.* 89: 645 - 655.

17. Kjellstrand, C.M., et al., 1971. Technical advances in hemodialysis of very small pediatric patients. *Proceedings of the Clinical Dialysis and Transplant Forum, 1971.* 124 - 132.

18. Goh, D. et al., 1994. The changing pattern of children's dialysis and transplantation over 20 years. *Clinical Nephrology.* 42:4 227 - 231.

19. Cameron, S., 1986. *Kidney Disease.* Oxford University Press.

20. Brown, E.R. and M.F. Epstein. Neonatal consequences of preterm birth. *In*: Fuchs, A., F. Fuchs and P. Stubblefield, 1993. *Preterm birth: causes, prevention and management.* 2nd ed. McGraw-Hill.

21. Coulthard, M. Renal function. *In*: Harvey, D., R. Cooke and G. Levitt, eds., 1989. *The baby under 1000g.* London: John Wright.

22. Brocklebank, J.T., 1988. Renal failure in the newly born. *Archives of Disease in Childhood.* 63: 991 - 994.

23. Bradbury, M.G., J.T. Brocklebank, E.H. Dyson, E. Goutcher and A.T. Cohen, 1995. Volumetric control of continuous haemodialysis in multiple organ failure. *Archives of Disease in Childhood.* 72: 42 - 45.

24. Donckerwolcke, R.A., and T.E. Bunchman, 1994. Hemodialysis in infants and small children. *Pediatric Nephrology.* 8:103 - 106.

25. Sadowski, R.H., W.E. Harmon and K. Jabs, 1994. Acute hemodialysis of infants weighing less than five kilograms. *Kidney International.* 45: 903 - 906.

26. Bock, G.H., A. Campos, T. Thompson, S. Michael Maher and C.M. Kjellstrand, 1981. Hemodialysis in the premature infant. *American Journal of Diseases of Children.* 135: 178 - 180.

27. Zobel, G., M. Kuttnig and E. Ring, 1990. Continuous arteriovenous hemodialysis in critically ill infants. *Child Nephrology and Urology.* 10: 196 - 198.

28. Forni, L.G. and P.J.Hilton, 1997. Continuous hemofiltration in the treatment of acute renal failure. *The New England Journal of Medicine.* 336: 1303 - 1309.

29. Ronco, C., 1994. Continuous renal replacement therapies in the treatment of acute renal failure in intensive care patients part 1. Theoretical aspects and techniques. *Nephrology Dialysis Transplantation.* 9: 191 - 200.

30. Henderson, L.W., E.A. Quellhorst, C.A. Baldamus and M.J.Lysaght, eds., 1986. *Hemofiltration.* Springer - Verlag.

31. Ronco, C., 1987. Acute renal failure in infancy. Treatment by continuous arterio-venous hemofiltration. *Proceedings of the 3rd International Symposium on Acute Renal Replacement Therapy, Florida 1987.* 111 - 134.

32. Lieberman, K.V., 1985. Treatment of acute renal failure in an infant using continuous arteriovenous hemofiltration. *The Journal of Pediatrics.* 106(4): 646 - 649.

33. Yorgin, P.D., A.M. Krensky and B.M. Tune,1990. Continuous venovenous hemofiltration. *Pediatric Nephrology.* 4: 640 - 642.

34. Klee, K.M., K. Greenleaf, L. Fouser and S.L. Watkins, 1996. Continuous venovenous hemofiltration with and without dialysis in pediatric patients. *ANNA Journal.* 23: 35 - 39.

35. Zobel, G., E. Ring and V. Zobel, 1989. Continuous arteriovenous renal replacement systems for critically ill children. *Pediatric Nephrology.* 3: 140 - 143.

36. Bunchman, T.E. and R.A. Donckerwolcke, 1994. Continuous arterial-venous diahemofiltration and continuous veno-venous diahemofiltration in infants and children. *Pediatric Nephrology.* 8: 96 - 102.

37. Steele, B.T., A. Vigneux, S. Blatz, M. Flavin and B. Paes, 1987. Acute peritoneal dialysis in infants weighing < 1500 g. *The Journal of Pediatrics.* 110: 126 - 129.

38. Summar, M., J. Pietsch, J. Deshpande and G. Schulman, 1995. Effective hemodialysis and hemofiltration driven by an extracorporeal membrane oxygenation pump in infants with hyperammonemia. *The Journal of Pediatrics.* 128: 379 - 382.

39. Besarab, A. and S. Frinak, 1998. The prevention of access failure: pressure monitoring. *ASAIO Journal.* 44: 35 - 37.

40. Besarab, A., T. Lubkowski, S. Frinak, S. Ramanathan and F. Escobar, 1997. Detecting vascular access dysfunction. *ASAIO Journal.* 43: M539 - M543.

41. Gotch, F.A., R. Buyaki, F. Panlilio and T. Folden, 1999. Measurement of blood access flow rate during hemodialysis from dialysance. *ASAIO Journal.* 45: 139 - 146.

42. Kupinsckas, R., H. Harjunmaa, S. Kun and R.A. Peura, 1998. Urea clearance monitoring during dialysis using an optical bridge method: feasibility studies. *Proceedings of the IEEE 24th Annual Northeast Bioengineering Conference.* 88 - 90.

43. Canaud, B., J.Y. Bosc, M. Leblanc, F. Vaussenat, H. Leray-Moragues, L.J. Garred, J.C. Mathieu-Daude and C. Mion, 1998. On-line dialysis quantification in acutely ill patients: preliminary clinical experience with a multipurpose urea sensor monitoring device. *ASAIO Journal.* 44: 184 - 190.

44. McMahon, M.P., S.B. Campbell, G.F. Shannon, J.S. Wilkinson and S.J. Fleming, 1996. A non-invasive continuous method of measuring blood volume during haemodialysis using optical techniques. *Medical Engineering and Physics.* 18: 105 - 109.

45. Maasrani, M., M.Y. Jaffrin and B. Boudailliez, 1997. Continuous measurements by impedance of haematocrit and plasma volume variations during dialysis. *Medical & Biological Engineering & Computing.* 35: 167 - 171.

46. Jaffrin, M.Y. and C. Fournier, 1999. Comparison of optical, electrical and centrifugation techniques for haematocrit monitoring of dialysed patients. *Medical & Biological Engineering & Computing.* 37: 433 - 439.

47. Lurzer, W., H. Scharfetter and H. Hutten, 1996. On-line blood and dialysate sampling system for multiparametric monitoring during dialysis. *18th Annual International Conference of the IEEE Engineering in Medicine and Biology Society, Amsterdam 1996.* 77 - 78.

48. Pallone, T.L., J. Petersen, 1987. A mathematical model of continuous arteriovenous hemofiltration predicts performance. *Transactions of the American Society for Artificial Internal Organs.* 33: 304 - 308.

49. Jenkins, R.D., J.E. Funk, B. Chen, T. Golper, 1992. A mathematical model for flow, pressure, and ultrafiltration rate in extracorporeal filtration of blood. *Blood Purification.* 10: 282 - 291.

50. Olbricht, C.J., M. Haubitz, U. Habel, U. Frei, K. Koch, 1990. Continuous arteriovenous hemofiltration: in vivo functional characteristics and its dependence on vascular access and filter design. *Nephron.* 55: 49 - 57.

51. Charm, S.E., G.S. Kurland, 1974. *Blood Flow and Microcirculation.* New York: John Wiley and Sons. pp. 30 - 36.

52. Landis, E.M., J.R. Pappenheimer, 1963. Exchange of substances through the capillary wall. *In:* Hamilton, W.F. ed., *Handbook of Physiology: Circulation.* Washington D.C.: American Physiology Society. pp. 962 - 1034.

53. Sargent, J.A., F.A. Gotch, 1978. Principles and biophysics of dialysis. *In:* Drukker, W., F.M. Parsons, J.F. Maher, eds., *Replacement of Renal Function by Dialysis.* The Hague: Nijhoff.

54. Pallone, T.L., S. Hyver, J. Petersen, 1989. The simulation of continuous arteriovenous hemodialysis with a mathematical model. *Kidney International.* 35: 125 - 133.

55. Akcahuseyin, E., H.H. Vincent, F.J. van Ittersum, W.A. van Duyl, M.A.D.H. Schalekamp, 1990. A mathematical model of continuous arterio-venous hemodiafiltration (CAVHD). *Computer Methods and Programs in Biomedicine.* 31: 215 - 224.

56. Pallone, T.L., S. Hyver, J. Petersen, 1988. Blood - dialysate equilibration during continuous arteriovenous hemodialysis. *Transactions of the American Society for Artificial Internal Organs.* 34: 512 - 514.

57. Akcahuseyin, E., W.A. van Duyl, M.C. Vos and H.H. Vincent, 1996. An analytical solution to solute transport in continuous arterio-venous hemodiafiltration (CAVHD). *Medical Engineering and Physics.* 18: 26 - 35.

58. Akcahuseyin, E., W.A. van Duyl, H.H. Vincent, M.C. Vos, M.A.D.H. Schalekamp, 1998. Continuous arterio-venous haemodiafiltration: hydraulic and diffusive permeability index of an AN-69 capillary haemofilter. *Medical and Biological Engineering and Computing.* 36: 43 - 50.

59. Vincent, H.H., F.J. van Ittersum, E. Akcahuseyin, M.C. Vos, W.A. van Duyl, M.A.D.H. Schalekamp, 1990. Solute transport in continuous arteriovenous hemodiafiltration: a new mathematical model applied to clinical data. *Blood Purification.* 8: 149 - 159.

60. Brunet, S., M. Leblanc, D. Geadah, D. Parent, S. Courteau, J. Cardinal, 1999. Diffusive and convective solute clearances during continuous renal replacement therapy at various dialysate and ultrafiltration flow rates. *American Journal of Kidney Diseases.* 34: 486 - 492.

61. Davenport, A., E.J. Will, A.M. Davison, 1990. Effect of the direction of dialysate flow on the efficiency of continuous arteriovenous haemodialysis. *Blood Purification.* 8: 329 - 336.

62. Winnett, R., J.C. James, T.C. Jannett, 1995. Simulation of blood volume changes during haemodialysis. *Proceedings of the IEEE Southeastcon 1995 New York.* 450 - 453.

63. Chamney, P.W., C. Johner, C. Aldridge, M. Kramer, N. Valasco, J.E. Tattersall, T. Aukaidey, R. Gordon, R.N. Greenwood, 1999. Fluid balance modelling in patients with kidney failure. *Journal of Medical Engineering and Technology.* 23: 45 - 52.

64. Sternby, J., 1998. Whole body urea kinetics. *Medical and Biological Engineering and Computing.* 36: 734 - 739.

65. Bolzern, P., P. Colaneri, D. Liberati, 1998. Robust estimation of urea concentration during dialytic treatment. *Proceedings of the 1998 IEEE International Conference on Control Applications, Trieste, Italy.* 75 - 79.

66. Vaussenat, F., J. Bosc, M. LeBlanc, B. Canaud, 1997. Data acquisition system for dialysis machines. A model for membrane hydraulic permeability. *ASAIO Journal.* 43: 910 - 915.

67. Julian D.G., 1984. *Cardiology.* 4th ed. Balliere Tindall.

68. Bengtsson, P. and J. Bosch, 1999. Haemodialysis software architecture design experiences. *Proceedings of the 1999 International Conference on Software Engineering, Los Angeles, California.* 516 - 525.

69. Sternby, J., 1996. Adaptive control of ultrafiltration. *IEEE Transactions on Control Systems Technology.* 4: 11 - 17.

70. Giove, S., M. Nordio and A. Zorat, 1993. An adaptive fuzzy control module for automatic dialysis. *Fuzzy Logic in Artificial Intelligence. 8th Austrian Artificial Intelligence Conference, 1993.* 146 - 156.

71. Giove, S., M. Nordio and B. Rossi, 1995. Fuzzy adaptive inference and medical decision support systems: a modular approach for dialysis procedure. *Proceedings of the WILF 1995 Italian Workshop on Fuzzy Logic.* 227 - 243.

72. Scharfetter, H., P. Bachhiesl, K. Kopke, F. Kappel and H. Hutten, 1996. Dynamical control of the dialysis process. Part I: structural considerations and first mathematical approach. *Biomedizinische Technik.* 41: 196 - 202.

73. Bachhiesl, P., H. Scharfetter, F. Kappel and H. Hutten, 1996. Dynamical control of the dialysis process. Part II: an improved algorithm for the solution of a tracking problem. *Biomedizinische Technik.* 41: 228 - 235.

74. Waugh, H.V. and A.J. Addlesee, 1997. A prototype model for renal vasculature and its application in haemodialyser miniaturisation. *Proceedings of the Institute of Mechanical Engineers.* 211: 479 - 482.

75. Waugh, H.V. and A.J. Addlesee, 1997. The feasibility of an artificial implantable kidney. *Proceedings of the 19th Annual International Conference of the IEEE Engineering in Medicine and Biology Society.* 2568 - 2571.

76. Mourad, G., R. Issautier and C. Mion, 1985. A portable hemodialysis system with sorbent regeneration of dialysate: preliminary results. *Transactions of the American Society for Artificial Internal Organs.* 31: 568 - 571.

77. Bigsby, R.J.A., R.J. Rider and G.N. Blount, 1998. The design of an optimized portable artificial kidney system using recirculation and regeneration of dialysate. *Proceedings of the Institute of Mechanical Engineers.* 212: 373 - 381.

78. De Virgiliis, G., M. Vanin and U. Buoncristiani, 1985. A new single needle dialysis system. *Transactions of the American Society for Artificial Internal Organs.* 31: 116 - 118.

79. De Wachter, D., P. Verdonck, R. Verhoeven and R. Hombrouckx, 1993. Comparison of a new and a standard single - needle dialysis system using a mathematical model. *Artificial Organs.* 17: 328 - 338.

80. Hombrouckx, R., A.M. Bogaert, F. Leroy, R. Beelen, J.Y. De Vos, G. Van Overmeeren, R. Verhoeven, P. Verdonck and V. Vercruysse, 1989. Limitations of short dialysis are the indications for ultrashort daily auto dialysis. *Transactions of the American Society for Artificial Organs.* 35: 503 - 505.

81. Roberts, M., and R.J. Winney, 1992. Errors in fluid balance with pump control of continuous hemodialysis. *The International Journal of Artificial Organs.* 15: 99 - 102.

82. Jenkins, R., H. Harrison, B. Chen, D. Arnold, and J. Funk, 1992. Accuracy of intravenous infusion pumps in continuous renal replacement therapies. *ASAIO Journal.* 38: 808 - 810.

83. Kameneva, M.V., J.F. Antaki, K.K. Yeleswarapu, M.J. Watach, B.P. Griffith, and H.S. Borovetz, 1997. *ASAIO Journal.* 43: M571 - M575.

84. Schaeffer, J., C.J. Olbricht, and K.M. Koch, 1996. Long - term performance of hemofilters in continuous hemofiltration. *Nephron.* 72: 155 - 158.

85. Ariadne 1 bidirectional blood pump publicity literature. Werken Glorieux, Renal Division, Ronse, Belgium.

86. Rosenson, R.S. et al., 1996. Distribution of blood viscosity values and biochemical correlates in healthy adults. *Clinical Chemistry.* 42:8 1189 - 1195.

87. Miniflow 10 hemofilter/dialyzer instructions for use. Hospal Renal Intensive Care.

88. Maplin catalogue, September 1998, p.658. Maplin Electronics Ltd.

89. *Medical electrical equipment - Part 1: General requirements for safety.* IEC 60601-1:1990. British Standards Institute, London.

90. *Medical electrical equipment - Part 2: Particular requirements for the safety of haemodialysis, haemodiafiltration and haemofiltration equipment.* IEC 60601-2-16: 1998. British Standards Institute, London.

91. Coulthard, M.G., 1985. Maturation of glomerular filtration in preterm and mature babies. *Early Human Development.* 11: 281 - 292.

92. Jaffrin, M.Y., B.B. Gupta and J.M. Malbrancq, 1981. A one-dimensional model of simultaneous hemodialysis and ultrafiltration with highly permeable membranes. *Journal of Biomedical Engineering.* 103: 261 - 266.

93. Schaefer, F., E. Straube, J. Oh, O. Mehls and E. Mayatepek, 1999. Dialysis in neonates with inborn errors of metabolism. *Nephrology, Dialysis and Transplantation.* 14: 910 - 918.

14. Bibliography

Andeen, G.B., 1988. *The Robot Design Handbook.* McGraw-Hill.

Auslander, D. and C. Kempf, 1996. *Mechatronics: mechanical system interfacing.* Prentice Hall.

Billingsley, J.B., 1989. *Controlling with Computers.* McGraw - Hill.

Briggs, J.D., 1994. *Renal Dialysis.* Chapman and Hall medical.

Dorf, R.C., 1989. *Modern Control Systems.* 5th ed. Addison - Wesley.

Harbison, S.P. and G.L. Steele, 1991. *C, A Reference Manual.* 3rd ed. Prentice - Hall.

Harvey, D., R. Cooke and G. Levitt, eds., 1989. *The Baby under 1000g.* London: John Wright.

Kernighan, B.W. and D.M. Ritchie, 1988. *The C Programming Language.* 2nd ed. Prentice - Hall.

Morris, N.M., 1991. *Control Engineering.* 4th ed. McGraw - Hill.

Tooley, M., 1995. *PC - based Instrumentation and Control.* 2nd ed. Butterworth - Heinemann.

Tortora, G.J. and S.R. Grabowski, 1992. *Principles of Anatomy and Physiology. 7th ed.* Harper Collins.

APPENDIX A Final Construction of Prototype

A.1 Construction of Electronic Circuitry

The electronic components were assembled on four single height Eurocard size matrix boards. Board 1 contains the pressure amplifier circuit, the stepper motor microswitch circuits and the 5 V - 12 V converters (figure A.1). Board 2 accommodates the circuitry for the 3 way tap drivers - the servo motor drivers and the photodetectors (figure A.2). Board 3 consists of the DAQ board interface circuit (figure A.3). And finally, board 4 is the DC - DC converter circuit (figure A.4). Standard 5 mm PCB mounted plugs and sockets were used to provide electrical connection to each board.

Matrix board was also used to mount the series resistors for the stepper motor coils. These were attached to the underside of the left and right base plates (figures A.5 and A.6). A further piece of matrix board was used to provide the wiring and interconnection for the photodetectors (figure A.5).

A.2 Photodetector Mountings

The photodetectors had to be mounted accurately and rigidly in position on the slotted disc. Matrix board was again used for this. Two small pieces were mounted at right angles on an aluminium pillar. The matrix board provides electrical connection as well as physical support. The pillar is attached to the servo motor studding by means of an aluminium plate. The whole assembly is shown in figure A.7.

A.3 Final Assembly of Electronics

All the circuit boards (apart from the DAQ interface board) were mounted in a half height Eurocard subrack (see figure A.8). This is bolted to the bottom of the case. The power supply is bolted to one of the end plates of the subrack. The DAQ interface board is attached to the front of the casing. The DAQ card ribbon cable socket is attached to the side of the casing and the mains power supply socket is at the rear. All interconnections between the various components are made with either 5 mm plugs and sockets or Molex microfit 3.0 connectors. The wiring is

designed so that any component can be easily unplugged and removed for fault finding and repair.

A.4 Casing

The casing was built using parts from the MB Building Kit System. This system allows the case to be built from preformed parts. The framework of the case was constructed from a 20 mm x 20 mm aluminium profile. Once the profiles had been cut to size they were simply bolted together using the MB fastening system. The overall dimensions of the case were 849 x 330 x 195 mm. The panelling was 2 mm aluminium sheet.

The lid was also constructed from 20 mm x 20 mm profile, and finished with 5 mm acrylic glass. The hinges used to connect the lid to the case were fitted with stops so that the lid could be held open in a stable position to allow easy access to the dialysis circuit (see figure A.9).

The dimensions of the case were determined by the size of the dialysis circuit. The haemofilter is relatively long at 30 cm, and this results in a large gap between the left and right aluminium base plates. This gap was covered with a 2 mm thick aluminium plate. It was necessary to provide an overlap between this plate and the base plates, to provide a degree of splash resistance to protect the electronics inside. The overlap was achieved by raising the base plates by 2 mm - a 2 mm frame was added to the underside of each base plate (see figures A.5 and A.6). The centre plate could now be slid forward to gain access to the inside of the case. The left, right and centre plates are all bolted to the top of the casing and can easily be removed for maintenance.

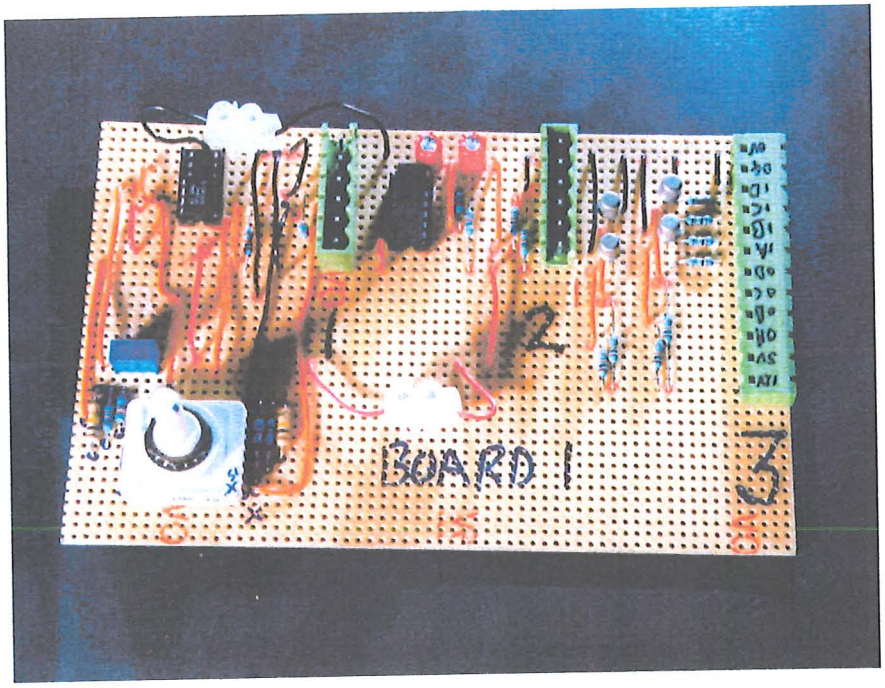


Figure A.1 Matrix Board 1

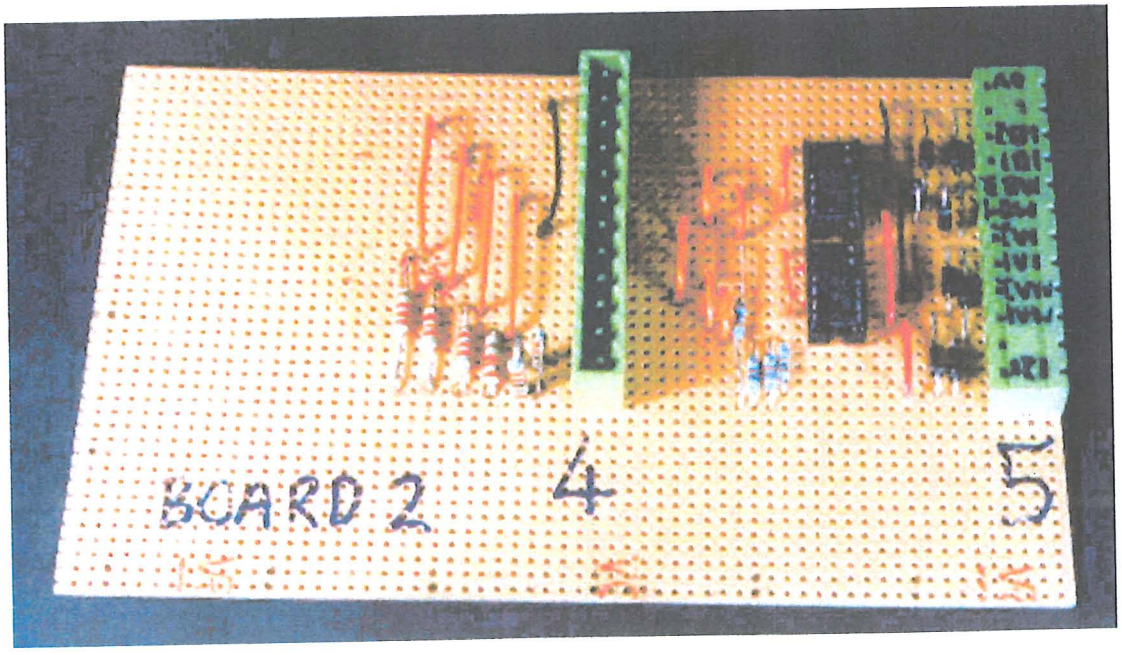


Figure A.2 Matrix Board 2

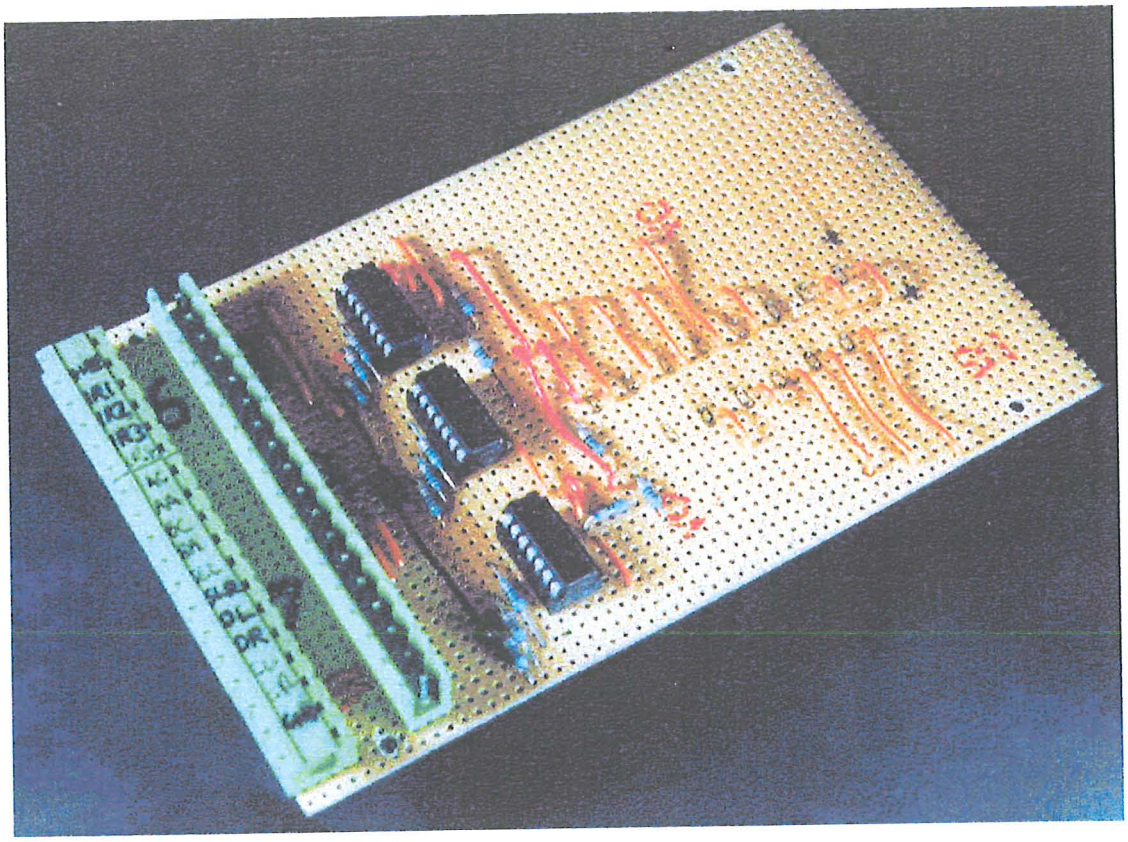


Figure A.3 DAQ Board Interface Circuit

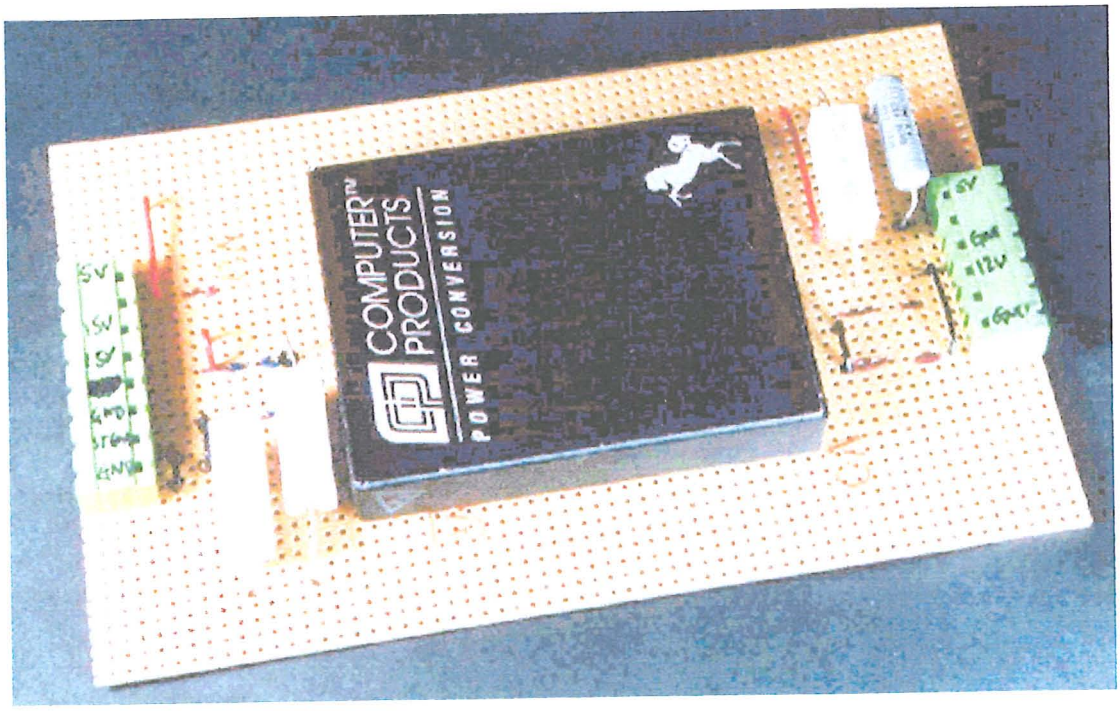


Figure A.4 DC - DC Converter Board

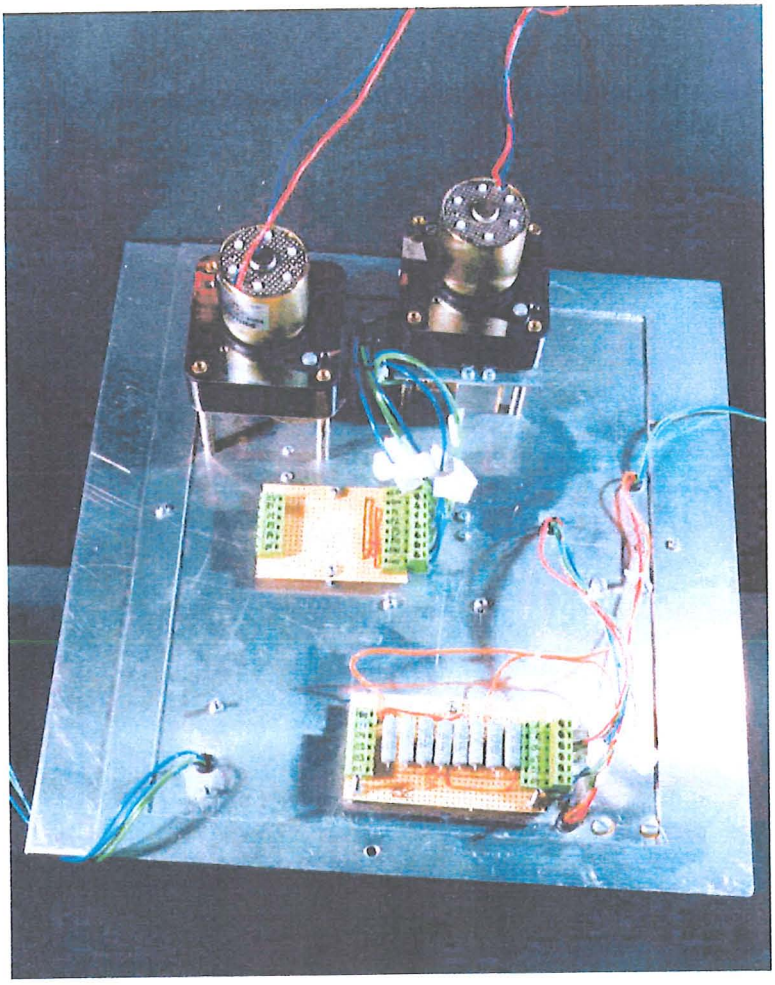


Figure A.5 Underside of Right Base Plate

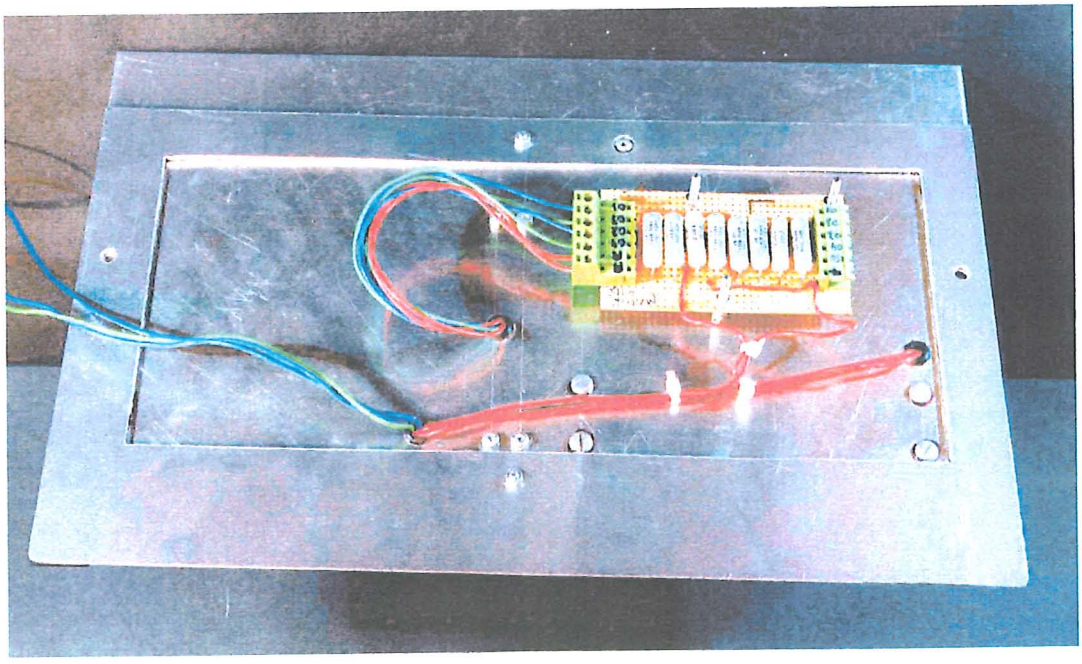


Figure A.6 Underside of Left Base Plate

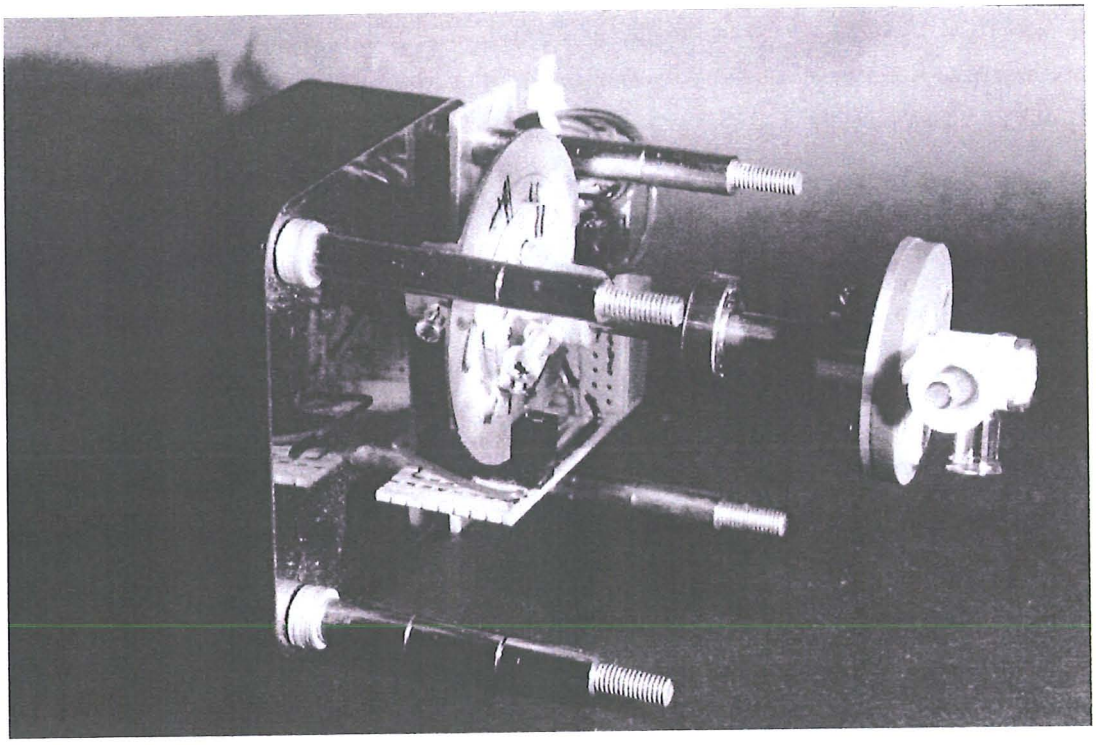


Figure A.7 Photodetector Mountings

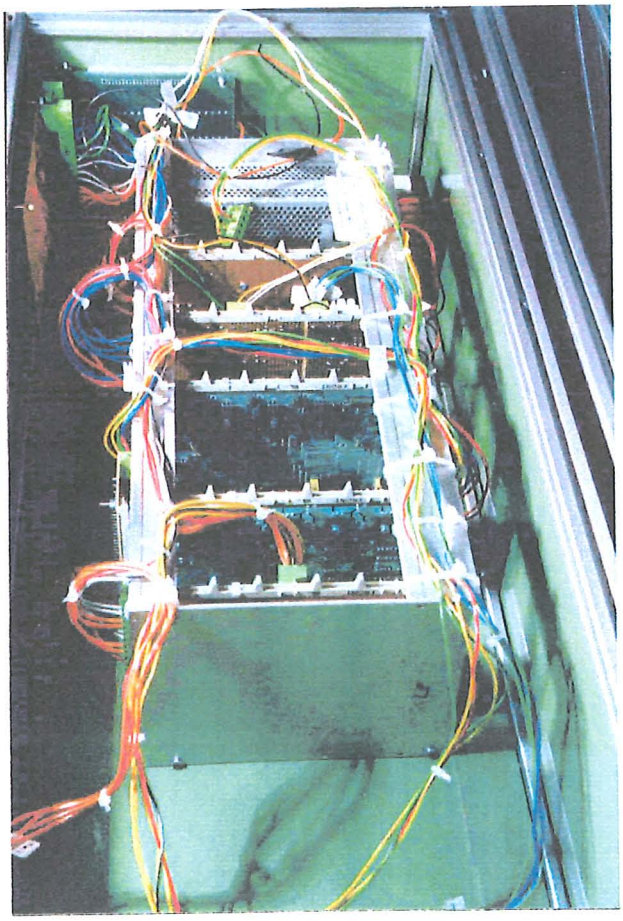


Figure A.8 Eurocard Subrack

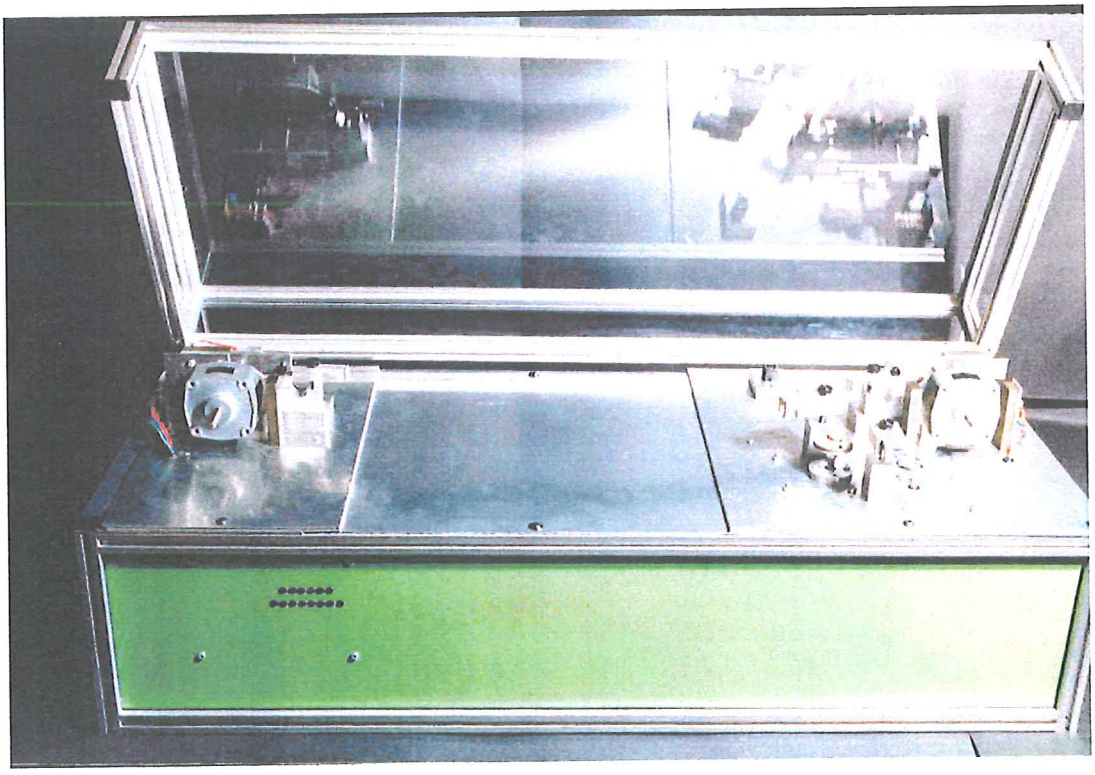
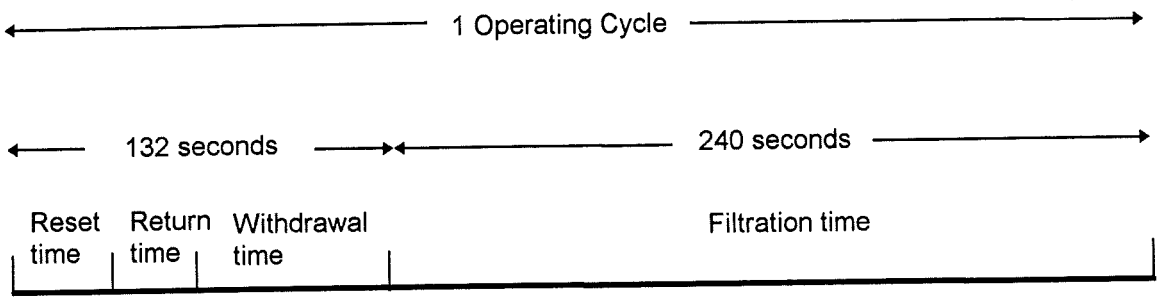


Figure A.9 Case

APPENDIX B Example of Filtration Rate Calculation



If mean filtration rate over 1 operating cycle = 5 ml/h

then actual filtration rate needed = $5 \times \left(\frac{132+240}{240}\right) = 7.75$ ml/h

APPENDIX C Complete Costing of Project

Haemodialysis Project - Complete Costing

Item	Cost (£)
200 MHz pentium PC - Gateway	1,838.00
DAQ card, cable, connector block - National Instruments	530.00
linear actuators - RS	157.73
stepper motor drive boards - RS	50.71
oscilloscope and power supply - John's Radio	420.00
Labwindows Upgrade - National Instruments	245.00
DC servo motors - Futaba	160.00
DC servo motors and gearboxes - McClellan	177.45
machining work	260.00
electronic components	800.37
mechanical components	615.89
miscellaneous items	105.26
Total =	5,360.41

APPENDIX D Program Listings

PUMP4.C

```
#include <header.h>

/* PUMP4.C
  FUNCTIONS IN THIS FILE:
  Filter_Blood(double f_rate,double* volume_filtered)
  Return_To_Start_Posn()

  this function controls ultrafiltration rate by moving the
  plungers at different speeds, rather than setting a TMP level and keeping
  it constant - remember this is open loop control

  FUNCTION SHOULD START WITH PLUNGER A DOWN AND PLUNGER B UP */

  /*****************************************************/

   int Filter_Blood(double f_rate, double* volume_filtered)

  /* rem order is board,port,line,input line state */

 {
  short push_switch_state,pull_switch_state,push_switch_line;
  short pull_switch_line,error_status,direction;
  int step_interval,syringe_pull,syringe_push,step_counter;
  int start_stroke_count, interval_counter = 0,filter_step_counter = 0;
  double,init_time,final_time,prev_f_step,current_f_step;
  double v_curr,v_ref,float_step_interval,mark,mark1;
 /* Set plunger movement directions
   order is: board,port,line,line state */

  init_time = Timer();

  direction = CLOCK;


 /* NB this is not being used at the moment */
  error_status = AI_VRead(1,0,2,&v_ref);
 /* if(v_curr < FILTER_LP_LIMIT || v_curr > FILTER_HP_LIMIT)
    return(10);    */

 /*work out the interval between filtering steps - work out the underlying
   flow rate in ml/hour and divide this by the filtration rate you want
   (also in ml/hour)*/

  float_step_interval  = (3600 * STEP_RATE/NUM_STEPS_PER_ML)/f_rate;

  step_interval = Round(float_step_interval);

  printf("step interval is %i\n",step_interval);

  printf("actual filtration rate: %6.2lf\n",f_rate);

 /*this outside loop cycles through the direction changes */


 step_counter = 0;
 mark = Timer();

 while(Timer() < init_time +  240.0)
```

```
59
60 {
61
62 /*printf("elapsed filtering time: %4.2lf secs\r",Timer()); this is only temporary*/
63 if(direction == CLOCK)
64
65  {/*set syringe A*/
66   DIG_Out_Line (1, 0, DIRN_A, UP);
67   /*set syringe B*/
68   DIG_Out_Line (1, 0, DIRN_B, DOWN);
69
70   syringe_pull = CLOCK_A;
71   syringe_push = CLOCK_B;
72
73   push_switch_line = SWITCH_B;
74   pull_switch_line = SWITCH_A;
75
76  }
77
78 else
79  {/*set syringe A*/
80   DIG_Out_Line (1, 0, DIRN_A, DOWN);       /* ie anticlockwise */
81   /*set syringe B*/
82   DIG_Out_Line (1, 0, DIRN_B, UP);
83
84   syringe_pull = CLOCK_B;
85   syringe_push = CLOCK_A;
86
87   push_switch_line = SWITCH_A;
88   pull_switch_line = SWITCH_B;
89
90  }
91
92
93 DIG_In_Line(1, 1, push_switch_line, &push_switch_state);
94 /*read push plunger switch before main loop starts*/
95
96 /* record step counter value at start of stroke - this enables the number of steps in */
97 /* each stroke to be monitored so failure testing can be done - remember the step counter */
98 /* variable is not reset except at the very start of this function */
99
100 start_stroke_count = step_counter;
101
102 /* Now enter the MAIN LOOP - this continues until plunger switch operates */
103
104
105 while(push_switch_state == 0 && Timer() < init_time + 240.0)
106 {
107
108  /* printf("%lf\n", Timer() - mark1);*/          /* these 2 lines print out the time taken*/
109  /* mark1 = Timer();          */                 /* by the basic step loop - if this goes above*/
110                                                  /* 0.025 sec then sync will be lost*/
111
112   SyncWait(mark,0.025 * step_counter);
113
114
115   GetUserEvent(0,&panel_event,&control_event);
116   if(global_stop == 1)
117    return(-1);
118
119   /* check for failure here */
120
121   if(step_counter - start_stroke_count > 2500)
```

```
122    return(8);
123
124   if(step_counter - start_stroke_count == 25)
125    if(pull_switch_state == 1)
126     return(9);
127
128  AI_VRead(1,0,2,&v_curr);
129  if(v_ref - v_curr > FILT_P_DROP || v_curr - v_ref > FILT_P_RISE)
130    return(10);
131
132
133
134
135   if(interval_counter == step_interval - 1)
136
137    /* this section steps just the push syringe */
138    {
139    DIG_Out_Line(1,0,syringe_push,0);
140    Delay(.0001);
141    DIG_Out_Line(1,0,syringe_push,1);
142    interval_counter = 0;
143    step_counter++;
144    filter_step_counter++;
145    }
146
147   else
148    /* this section steps both plungers */
149    {
150    DIG_Out_Line(1,0,syringe_push,0);
151    DIG_Out_Line(1,0,syringe_pull,0);
152    Delay(.0001);
153    DIG_Out_Line(1,0,syringe_push,1);
154    DIG_Out_Line(1,0,syringe_pull,1);
155    interval_counter++;
156    step_counter++;
157    }
158
159
160
161  DIG_In_Line(1, 1, push_switch_line, &push_switch_state); /*read push plunger switch*/
162  DIG_In_Line(1, 1, pull_switch_line, &pull_switch_state);
163 }
164
165 /* if(no_of_step >   then return */
166
167 direction = !direction;
168
169 }
170
171
172 *volume_filtered =(double)filter_step_counter/NUM_STEPS_PER_ML;
173 /*calculates the volume filtered in ml - each step is 1/246 ml (with a BD 10ml syringe)*/
174
175
176 printf("step counter = %i\n",step_counter);
177
178 /* allow 5 steps for errors at the end - should be 9600 strictly*/
179 if(step_counter < 9595)
180   return(21);
181
182 return(0);
183
184}
```

```
185
186
187
188 /*************************************************************/
189
190
191
192
193 /* note that if plunger A is already down then nothing will happen - so plunger B can be
194    in any position - so starting with both down is OK */
195
196
197 int Return_To_Start_Posn()
198
199 {
200 short push_switch_line,push_switch_state;
201 int step_counter = 0;
202 double mark,v_ref,v_curr;
203
204
205
206 push_switch_line = SWITCH_A;
207
208 /*set syringe A*/
209 DIG_Out_Line (1, 0, DIRN_A, DOWN);        /* ie anticlockwise */
210 /*set syringe B*/
211 DIG_Out_Line (1, 0, DIRN_B, UP);
212
213
214 AI_VRead(1,0,2,&v_ref);
215
216 mark = Timer();
217
218 DIG_In_Line(1, 1, push_switch_line, &push_switch_state); /*read plunger switch*/
219
220 while(push_switch_state == 0)
221
222 {
223  SyncWait(mark,0.025 * step_counter);
224
225
226   GetUserEvent(0,&panel_event,&control_event);
227   if(global_stop == 1)
228     return(-1);
229
230
231
232
233  DIG_Out_Line(1,0,CLOCK_A,0);
234  DIG_Out_Line(1,0,CLOCK_B,0);
235  Delay(.0001);
236  DIG_Out_Line(1,0,CLOCK_A,1);
237  DIG_Out_Line(1,0,CLOCK_B,1);
238  step_counter++;
239  DIG_In_Line(1, 1, push_switch_line, &push_switch_state); /*read plunger switch*/
240
241  /* this bit checks that the plunger hasn't gone too far - i.e. checks that
242     steppers are working and also the microswitches */
243  if(step_counter > NO_OF_STEPS_LIMIT)
244   return(4);
245
246  /* check pressure and trigger alarm if it's outside the safety limits - these limits
247     are the same as in Filter_Blood() */
```

```
248 AI_VRead(1,0,2,&v_curr);
249 if(v_ref - v_curr > FILT_P_DROP || v_curr - v_ref > FILT_P_RISE)
250   return(11);
251 }
252 return(0);
253 }
```

WITHRET.C

```
1
2 /*  WITHRET.C   ---- this file contains two functions - one for
3   withdrawing blood and one for returning it */
4
5 /*FUNCTIONS IN THIS FILE: */
6       /*  Withdraw_Blood()
7         Return_Blood()
8         Reverse_Flow()
9                                                 */
10
11
12 #include <Header.h>
13
14
15 /*****************************************************************/
16 int Withdraw_Blood(double blood_volume)
17
18 {
19
20  short switch_state,reverse_flow_count = 0,access_mode;
21  int delay_counter=0, step_counter=0, num_steps,return_code;
22  double mark, start_time,v_ref, v_curr,time_limit,speed_factor = 1.0;
23
24 /* note: stepper delay is multiplied by speed factor - so the larger the speed factor, the slower
25    the motor goes */
26
27
28 /* get access mode from the user interface */
29
30 GetCtrlVal(handle1, PANEL_ACCESS_MODE, &access_mode);
31
32
33 AI_VRead(1,0,2,&v_ref);
34
35 /* this code tests that plunger is at bottom of syringe and also that microswitch
36    hasn't failed to the zero state */
37
38 DIG_In_Line(1, 1, SWITCH_B, &switch_state); /*read plunger switch*/
39 if(switch_state == 0)
40   return(6);
41
42
43 /*set syringe B to travel upwards*/
44
45 DIG_Out_Line (1,0,DIRN_B, UP);
46
47
48 num_steps = Round(blood_volume * NUM_STEPS_PER_ML);
49
50 /* this line sets a time limit on blood withdrawal proportional to the volume to be withdrawn*/
51
52 time_limit = blood_volume * VOL_TIME_FACTOR;
```

```
53
54
55 mark = Timer(); /* this is the first call to the timer */
56
57 /* this loop tests to see if the right volume has been reached yet
58   and also if the whole process has taken too long*/
59
60
61
62 /* BEGINNING OF MAIN LOOP */
63
64 while(step_counter < num_steps && Timer() < mark + time_limit)
65 {
66
67  /* test to see if button has been pressed */
68
69  GetUserEvent(0,&panel_event,&control_event);
70   if(global_stop == 1)
71    return(-1);
72
73  /* test here to make sure microswitch B hasn't failed to 1 state */
74  /* the value of 20 is fairly arbitrary - just needs to be enough */
75  /* to make the switch change state */
76
77  if(step_counter == 20)
78   {DIG_In_Line(1, 1, SWITCH_B, &switch_state); /*read plunger switch*/
79    if(switch_state == 1)
80     return(7);
81   }
82
83
84  AI_VRead(1,0,2,&v_curr);
85  if(v_ref - v_curr > WITHRET_P_DROP || v_curr - v_ref > WITHRET_P_RISE)
86   return(13);
87
88  /* figure of 1.3 v based on voltage drop with 0.5 ml volume change with tube clamped 10cm
89    along thin section*/
90  /* 0.05 added to allow for 'relaxation' once the 1st 1 second delay has been triggered -
91    even if pressure has risen a bit delay will still be triggered again - value 0.05 is fairly
92    arbitrary - it can be changed if another value works better*/
93
94  if((v_ref - v_curr > WITH_DELAY_LIMIT && delay_counter == 0) ||
95  (v_ref - v_curr > WITH_DELAY_LIMIT - 0.05 && delay_counter > 0))
96
97
98    if(reverse_flow_count >1 && access_mode == 2)
99     {
100      Delay(2.0);
101      speed_factor = speed_factor * 2.0;
102      printf(" speed factor = %2.1f\n",speed_factor);
103      reverse_flow_count = 0;
104      Delay(5.0);
105     }
106    else
107     {
108              Delay(1.0);
109              delay_counter++;
110              printf("delay counter = %i\n",delay_counter);
111              if(delay_counter > 4)
112              {
113                       return_code = Reverse_Flow(&step_counter,v_ref);
114                       reverse_flow_count++;
115                       printf("finished reverse flow\n");
```

```
116                     Delay(2.0);       /* not sure whether to keep this or not */
117             if(return_code != 0)
118               return(return_code);
119             delay_counter = 0;
120         }
121     }
122
123
124 else
125
126   { /* this section steps the B syringe */
127     DIG_Out_Line(1,0,CLOCK_B,0);
128     Delay(.0001 * speed_factor);
129     DIG_Out_Line(1,0,CLOCK_B,1);
130     Delay(.0249 * speed_factor);
131     step_counter++;
132     delay_counter = 0;
133   }
134
135
136
137
138 }               /* END OF MAIN LOOP */
139
140
141
142 if(Timer() - mark > time_limit)
143   {printf("time limit = %lf\n",time_limit);
144    return(2);
145   }
146 else
147   return(0);
148
149 }
150
151
152
153
154
155
156
157
158 /*****************************************************************/
159
160 int Return_Blood()
161
162{
163
164 short switch_state;
165 int step_counter=0;
166 double mark, v_ref,v_curr;
167
168 DIG_In_Line(1, 1, SWITCH_B, &switch_state); /*read plunger switch*/
169 DIG_Out_Line (1,0,DIRN_B, DOWN);  /*set syringe B to move down*/
170
171 AI_VRead(1,0,2,&v_ref);
172
173 mark = Timer(); /* this is the first call to the timer */
174 while(switch_state == 0)
175 {
176
177   GetUserEvent(0,&panel_event,&control_event);
178   if(global_stop == 1)
```

```
179    return(-1);
180
181
182    SyncWait(mark,0.025 * step_counter);
183
184   /* this section steps the B syringe */
185
186    DIG_Out_Line(1,0,CLOCK_B,0);
187    Delay(.0001);
188    DIG_Out_Line(1,0,CLOCK_B,1);
189    step_counter++;
190    DIG_In_Line(1, 1, SWITCH_B, &switch_state); /*read plunger switch*/
191
192     /* this bit checks that the plunger hasn't gone too far - i.e. checks that
193    steppers are working and also the microswitches */
194      if(step_counter > NO_OF_STEPS_LIMIT)
195      return(5);
196
197    /* check pressure and trigger alarm if outside safety limits */
198
199     AI_VRead(1,0,2,&v_curr);
200     if(v_ref - v_curr > WITHRET_P_DROP || v_curr - v_ref > WITHRET_P_RISE)
201       return(12);
202
203
204
205
206
207 }
208 /*temporary bit to count steps*/
209 printf("no. of steps(return): %i \n",step_counter);
210
211 return(0);
212 }
213
214 /*********************************************************************/
215
216
217 /*note that value of v_ref is passed in from withdraw_blood(), not read
218   inside this function, as function is not called at ambient pressure */
219
220 int Reverse_Flow(int *step_counter,double v_ref)
221
222 {
223
224  short reverse_counter=0, switch_state;
225  double v_curr;
226
227
228  DIG_Out_Line (1,0,DIRN_B, DOWN);  /*set syringe B to move down*/
229  DIG_In_Line(1, 1, SWITCH_B, &switch_state); /*read plunger switch*/
230
231  printf("reversing flow....\n");
232 /*this is set to 123 steps at the moment - this is 0.5 ml*/
233
234 while(reverse_counter < 123 && switch_state == 0)
235
236      {
237              DIG_Out_Line(1,0,CLOCK_B,0);
238       Delay(.0125);
239       DIG_Out_Line(1,0,CLOCK_B,1);
240       Delay(.0125);
241              (*step_counter)--;
```

```
242             reverse_counter++;
243       DIG_In_Line(1, 1, SWITCH_B, &switch_state); /*read plunger switch*/
244
245       /* test pressure */
246
247        AI_VRead(1,0,2,&v_curr);
248        if(v_ref - v_curr > WITHRET_P_DROP || v_curr - v_ref > WITHRET_P_RISE)
249          return(14);
250      }
251 DIG_Out_Line (1,0,DIRN_B, UP);  /*set syringe B to move up again*/
252 return(0);
253 }
```

SERVO2.C

```
1 /*                                *****SERVO2.C*******
2
3 FUNCTIONS IN THIS FILE:
4
5       MoveServo()
6       Current_Posn()
7       Change_Direction()
8       Check_Servo_Posn()
9
10
11
12  THIS IS THE NEW VERSION OF THE SERVO DRIVING PROGRAM.
13  BECAUSE OF PROBLEMS WITH THE OLD VERSION DETECTING SLOT
14  EDGES THE HARDWARE HAS BEEN CHANGED - NOW THERE IS ONE
15  SLOT AND TWO DETECTORS - ONE FOR EACH POSITION. SO ONLY
16  TWO POSITIONS CAN BE MOVED TO INSTEAD OF THE PREVIOUS
17  THREE.
18
19 */
20
21 #include <header.h>
22
23 /*****************************************************/
24
25 int MoveServo(int Servo_Num,int Target_Posn)
26
27 {
28 int difference,i_line1,i_line2,pulse_counter=0,cycle_counter=0;
29 int no_movement =  0,state_11,return_code;
30 short output_line;
31
32
33 /* first find current position of servo
34  and therefore calculate which direction servo needs to move */
35
36 switch(Servo_Num)
37
38 {case 1:
39                 i_line1 = A1;
40                 i_line2 = A2;
41
42
43
44 /* Now calculate direction that you need to move to get from the current
45  position to the target position */
46
```

```
47      difference = Target_Posn - Current_Posn(i_line1,i_line2);
48      if(difference > 0) output_line = CLOCK_1;
49      if(difference < 0) output_line = ANTI_1;
50
51      /* this is needed for no movement position check */
52      if(difference == 0)output_line = ANTI_1;
53
54
55              break;
56 case 2:
57              i_line1 = B1;
58              i_line2 = B2;
59
60
61
62
63      difference = Target_Posn - Current_Posn(i_line1,i_line2);
64      if(difference > 0) output_line = CLOCK_2;
65      if(difference < 0) output_line = ANTI_2;
66
67      /* as above, needed for no movement position check */
68      if(difference == 0)output_line = ANTI_2;
69
70
71              break;
72 }
73
74
75
76 /* Now move the servo in the direction worked out above to get to the
77   target position.
78   remember argument order is board,port,line,port width,configure,line state*/
79
80 state_11 = 0;
81
82 if(Current_Posn(i_line1,i_line2) == Target_Posn)
83   no_movement = TRUE;
84
85
86 while(Current_Posn(i_line1,i_line2) != Target_Posn )
87   {while(Current_Posn(i_line1,i_line2) != Target_Posn )
88      {
89       DIG_Out_Line (1, 0, output_line,1);
90       Delay(.019);
91       DIG_Out_Line (1, 0, output_line,0);
92
93       /* this bit is to catch any line failing to zero */
94       if(state_11 == 0)
95         if(Current_Posn(i_line1,i_line2) == 3)
96           state_11 = 1;
97       /* end of this bit */
98       pulse_counter++;
99       if(pulse_counter > 1000) /* was 150 */
100        break;
101      }
102    pulse_counter = 0;
103    Change_Direction(&output_line);
104    cycle_counter++;
105    if(cycle_counter > 8)   /* was 5 */
106      break;
107   }
108
109 if(cycle_counter > 8)
```

```
110  return(1);
111 if(state_11 == 0 && no_movement == FALSE)
112  return(3);
113
114 /* now check servo position if there's been no movement so far */
115
116
117 if(no_movement)
118  {return_code = Check_Servo_Posn(i_line1,i_line2,output_line);
119   if(return_code)
120   return(return_code);
121  }
122
123 return(0);              /*normal return with no errors*/
124 }
125
126
127 /*****************************************************/
128
129 int Current_Posn(int i_line1,int i_line2)
130
131 {
132  int Current_Pos = 0;
133  short state1,state2;
134
135 DIG_In_Line (1, 1, i_line1, &state1);
136 DIG_In_Line (1, 1, i_line2, &state2);
137 if(state1==0 && state2==1) Current_Pos = 1;
138 if(state1==1 && state2==0) Current_Pos = 2;
139 if(state1==1 && state2==1) Current_Pos = 3;      /*intermediate state*/
140
141 return(Current_Pos);
142 }
143
144
145 /*****************************************************/
146
147 int Change_Direction(short *o_line)
148  {
149   switch(*o_line)
150   {      case 4:  *o_line = 5;
151                          break;
152
153      case 5:  *o_line = 4;
154                 break;
155
156      case 6:  *o_line = 7;
157                          break;
158
159      case 7:  *o_line = 6;
160                 break;
161    }
162    return(0);
163   }
164
165
166 /*****************************************************/
167
168
169 int Check_Servo_Posn(int i_line1,int i_line2,short output_line)
170 {
171
172  int pulse_counter = 0,start_posn;
```

```
173
174  start_posn = Current_Posn(i_line1,i_line2);
175
176
177  while(Current_Posn(i_line1,i_line2) != 3 )
178      {
179       DIG_Out_Line (1, 0, output_line,1);
180       Delay(.02);
181       DIG_Out_Line (1, 0, output_line,0);
182       pulse_counter++;
183       if(pulse_counter > 150)
184         return(3);
185      }
186
187
188  Change_Direction(&output_line);
189  pulse_counter = 0;
190
191  while(Current_Posn(i_line1,i_line2) != start_posn )
192      {
193       DIG_Out_Line (1, 0, output_line,1);
194       /*WriteToDigitalLine (1, "0", output_line, 8, 0, 1); */
195       Delay(.02);
196       DIG_Out_Line (1, 0, output_line,0);
197       /*WriteToDigitalLine (1, "0", output_line, 8, 0, 0);*/
198       pulse_counter++;
199       if(pulse_counter > 150)
200         return(3);
201       }
202
203  return(0);
204 }
```

RUNSYSTEM.C

```
1 /*****************************RunSystem.c*****************************/
2
3 /*  FUNCTION LIST:
4
5   run_system()    */
6
7 #include <Header.h>
8
9
10
11 int run_system(double blood_volume,double mean_f_rate)
12
13 {
14
15  int return_code;
16
17  double f_rate,with_time,res_time,ret_time,filter_time,off_time_start,off_time;
18  double volume_filtered = 0, cum_f_volume = 0;
19  double start_time;
20
21  global_stop = 0;
22
23 /* reset the screen timer every time the system is restarted */
24 start_time = Reset_Timer();
25
26 /* move taps into reset position for first pass through the loop */
```

```
27
28 return_code = MoveServo(1,2);
29 if(return_code != 0)
30   return(return_code);
31
32
33 return_code = MoveServo(2,1);
34 if(return_code != 0)
35   return(return_code);
36
37
38 /* MAIN LOOP STARTS HERE */
39
40  while(TRUE)
41
42 {
43
44 /* take time at start of cycle */
45 off_time_start = Timer();
46
47 /*this returns if the stop button has been pressed while inside Filter_Blood()*/
48 if(return_code != 0)
49   return(return_code);
50
51 /* update timer display at this point - it doesn't really matter at what point
52    in the cycle this is done */
53 Update_Timer(start_time);
54
55 /* now update cumulative filtrate display */
56
57 cum_f_volume = cum_f_volume + (volume_filtered/FILT_CORRECTION_FACTOR);
58
59 SetCtrlVal(handle1,PANEL_CUM_VOLUME,cum_f_volume);
60 ProcessSystemEvents();            /*added 19/1/99 */
61
62
63 /* RESET SYRINGES - move syringes to A down and B up - NB this will do
64    nothing on first pass */
65
66 printf("resetting....\n");
67 return_code = Return_To_Start_Posn();
68 if(return_code != 0)
69   return(return_code);
70
71
72
73 /* delay in here to allow pressure gradient in filter to smooth out at end of reset */
74
75 Delay(5);
76
77
78
79 /* MOVE SERVOS TO WITHDRAW AND RETURN POSITION */
80
81 return_code = MoveServo(1,1);
82 if(return_code != 0)
83   return(return_code);
84
85 return_code = MoveServo(2,2);
86 if(return_code != 0)
87   return(return_code);
88
89
```

```
90
91  /* RETURN BLOOD - again this will do nothing on first pass */
92
93  printf("returning blood...\n");
94  return_code = Return_Blood();
95  if(return_code != 0)
96    return(return_code);
97
98
99
100 /* put in a delay to allow pressure to rise to ambient */
101
102 Delay(3.0);
103
104 /* WITHDRAW BLOOD */
105
106 return_code = Withdraw_Blood(blood_volume);
107 if(return_code != 0)
108   return(return_code);
109
110 printf("withdraw time: %6.2lf\n",with_time);
111
112 /*another delay for pressure equalization */
113
114 Delay(4.0);
115
116 /* MOVE TAPS TO FILTERING POSITION */
117  return_code = MoveServo(2,1);
118 if(return_code != 0)
119   return(return_code);
120
121 return_code = MoveServo(1,2);
122 if(return_code != 0)
123   return(return_code);
124
125
126
127 /* CALCULATE FILTRATION RATE NEEDED */
128
129 off_time = Timer() - off_time_start;
130 printf("total off time: %6.2lf\n",off_time);
131
132 f_rate = (mean_f_rate/240)*(240.0 + off_time);
133
134
135
136 /* BEGIN FILTRATION */
137
138
139
140 return_code = Filter_Blood(f_rate,&volume_filtered);
141 /* see top of loop for return_code bit */
142
143 /* end filtration */
144
145 /*printf("filtration time: %6.2lf\n",filter_time);          */
146
147
148 }    /*END OF MAIN LOOP HERE*/
149
150
151
152 /* now do final update of cumulative filtrate display */
```

```
153
154 cum_f_volume = cum_f_volume + (volume_filtered/FILT_CORRECTION_FACTOR);
155 /* correcting volume back again so screen reflects actual target
156    volume not corrected target volume*/
157 SetCtrlVal(handle1,PANEL_CUM_VOLUME,cum_f_volume);
158 Update_Timer(start_time);
159 ProcessSystemEvents();
160
161 return(0);
162 }
```

HAEMO1.C

```
1 /* HAEMO1.C */
2
3 /* FUNCTIONS IN THIS FILE: */
4
5
6 /*     main()
7       system_init()
8       system_start()
9       change_filter_rate()
10      system_stop()
11      quit_system()      */
12
13
14
15 #include <userint.h>
16 #include <ansi_c.h>
17 #include <analysis.h>
18 #include <utility.h>
19 #include <Haemo1.h>
20 #include <header.h>
21
22
23
24
25
26
27 main()
28 {
29      int month,day,year;
30      char date_string[9];
31      char* time_string;
32      char string[100];
33      char year_string[3];
34      char whole_year_string[5];
35
36      DIG_Prt_Config (1, 0, 0, 1);  /* Configures port 0 for output */
37
38      DIG_Prt_Config (1, 1, 0, 0);  /* Configures port 1 for input */
39
40
41   /* set the init parameters to zero */
42
43   software_init = 0;
44   hardware_init = 0;
45
46      handle1 = LoadPanel (0, "haemo1.uir", PANEL);
47      handle2 = LoadPanel (handle1, "haemo1.uir", RESPONSE);
48      handle3 = LoadPanel (handle1, "haemo1.uir", ALARM);
```

```
InsertTextBoxLine (handle1,PANEL_MESSAGE_BOX,0,"");
InsertTextBoxLine (handle1,PANEL_MESSAGE_BOX,1,"FOLLOW ALL THE
                    INSTRUCTIONS GIVEN EXACTLY.");
InsertTextBoxLine (handle1,PANEL_MESSAGE_BOX,2,"");
InsertTextBoxLine (handle1, PANEL_MESSAGE_BOX, 3,"Before you can start
                    the treatment session");
InsertTextBoxLine (handle1, PANEL_MESSAGE_BOX, 4,"you must initialise the system.
                    Press the");
InsertTextBoxLine (handle1,PANEL_MESSAGE_BOX, 5,"button at the top left of the screen.");
    DisplayPanel (handle1);
    SetCtrlAttribute (handle1,PANEL_START,ATTR_DIMMED,1);

    SetCtrlAttribute (handle1,PANEL_CHANGE_RATE,ATTR_DIMMED,1);
    SetCtrlAttribute (handle1,PANEL_STOP,ATTR_DIMMED,1);

    SetCtrlAttribute (handle1,PANEL_BIRTH_WEIGHT,ATTR_DIMMED,1);



    /* initialise the treatment history box */
    GetSystemDate(&month,&day,&year);
    Fmt(whole_year_string,"%s<%i",year);
    Fmt(year_string,"%s[i2]",whole_year_string);
    Fmt(date_string,"%i%s%i%s%s",day,"/",month,"/",year_string);
    time_string = TimeStr();
    Fmt(string,"%s%s%s%s[w5]%s","Start:  ",date_string,"   ",time_string,"\n\n");
    SetCtrlVal(handle1,PANEL_HISTORY,string);


  RunUserInterface();




}

/*****************************************************************************/

int CVICALLBACK system_init(int panel, int control, int event, void *callbackData,
                            int eventData1, int eventData2)
{
    int return_code;




if(event==EVENT_COMMIT)
 {

   SetCtrlAttribute (handle1,PANEL_INITIAL,ATTR_DIMMED,1);
   SetCtrlAttribute (handle1,PANEL_START,ATTR_DIMMED,1);
   SetCtrlAttribute (handle1,PANEL_STOP,ATTR_DIMMED,1);
   SetCtrlAttribute (handle1,PANEL_CHANGE_RATE,ATTR_DIMMED,1);
   SetCtrlAttribute (handle1,PANEL_QUIT,ATTR_DIMMED,1);
    .


  if(hardware_init == TRUE)
    {return_code = GenericMessagePopup("","Do you want to reinitialise everything or just the
     treatment parameters?","Just reenter the treatment parameters","Reinitialise everything",
     0,0,1,VAL_GENERIC_POPUP_BTN1,VAL_GENERIC_POPUP_NO_CTRL,
```

```
112     VAL_GENERIC_POPUP_NO_CTRL,0);
113       if(return_code == 2)
114               hardware_init = FALSE;
115     }
116  /*   initialise hardware here  */
117
118
119  if(hardware_init == FALSE)
120    { return_code = init_hardware();
121      if(return_code > 0)
122       {process_error(return_code);
123        return(1);
124       }
125      if(return_code == 0)
126        hardware_init = TRUE;
127
128    }
129
130
131
132     /* this return means that software isn't initialised if hardware fails */
133
134  do
135    init_software();
136  while(!ConfirmPopup("Confirm Parameters","Do you want
137                                to proceed with these parameters?"));
138  software_init = TRUE;
139
140
141  /*now parameters have been set, update the screen to be ready to start the system */
142
143    ResetTextBox(handle1, PANEL_MESSAGE_BOX, "");
144    InsertTextBoxLine(handle1,PANEL_MESSAGE_BOX,0,"The system is now initialised with
145                              the parameters ");
146    InsertTextBoxLine(handle1,PANEL_MESSAGE_BOX,1," shown in the boxes below.");
147    InsertTextBoxLine(handle1,PANEL_MESSAGE_BOX,2," Press the 'start treatment' button
148                              when you are ready.");
149
150     SetCtrlAttribute (handle1, PANEL_INITIAL, ATTR_DIMMED, 0);
151     SetCtrlAttribute (handle1, PANEL_START, ATTR_DIMMED, 0);
152     SetCtrlAttribute (handle1,PANEL_CHANGE_RATE,ATTR_DIMMED,0);
153     SetCtrlAttribute (handle1, PANEL_QUIT, ATTR_DIMMED, 0);
154
155     /* take the focus away from the filtration rate control */
156     SetActiveCtrl(handle1,PANEL_START);
157
158
159
160
161 }
162 return(0);
163 }
164
165
166 /*****************************************************************************/
167     .
168
169 int system_start(int panel, int control, int event, void *callbackData,
170                          int eventData1, int eventData2){
171
172  double blood_volume,filter_rate, corrected_filter_rate;
173  char start_time[6],finish_time[6],string[50];
174  int hours,minutes,return_code;
```

```
175 double cum_volume;
176
177
178
179 if (event==EVENT_COMMIT)
180    {
181     SetCtrlAttribute (handle1,PANEL_CHANGE_RATE,ATTR_DIMMED,0);
182     SetCtrlAttribute (handle1, PANEL_STOP, ATTR_DIMMED, 0);
183     SetCtrlAttribute (handle1, PANEL_QUIT, ATTR_DIMMED, 1);
184     SetCtrlAttribute (handle1,PANEL_START,ATTR_DIMMED,1);
185     SetCtrlAttribute (handle1,PANEL_INITIAL,ATTR_DIMMED,1);
186
187     SetCtrlVal(handle1,PANEL_RUNNING,1);
188     SetCtrlVal(handle1,PANEL_STOPPED,0);
189
190     ResetTextBox(handle1, PANEL_MESSAGE_BOX, "");
191     InsertTextBoxLine(handle1,PANEL_MESSAGE_BOX,0," The system is running.");
192
193     /* better to get the values 'straight off the screen' rather than have
194       globals containing them */
195
196     GetCtrlVal(handle1,PANEL_WB_VOLUME,&blood_volume);
197     GetCtrlVal(handle1,PANEL_FILTER_RATE,&filter_rate);
198
199     Fmt(start_time,"%s[w5]",TimeStr());
200
201     /*PUT IN CORRECTION FACTOR HERE (18/1/99)*/
202
203     corrected_filter_rate = FILT_CORRECTION_FACTOR * filter_rate;
204
205     return_code = run_system(blood_volume,corrected_filter_rate);
206                                          /* this function is in RunSystem.c */
207     /* only process an error if code is greater than zero - if stop button
208       is pressed the return code is -1. Then the code just drops through which
209       is OK */
210
211     if(return_code > 0)
212      process_error(return_code);
213
214     Fmt(finish_time,"%s[w5]",TimeStr());
215     Fmt(string,"%s%s%s%s",start_time," to ",finish_time,"\n");
216     SetCtrlVal(handle1,PANEL_HISTORY,string);
217     GetCtrlVal(handle1,PANEL_HOURS,&hours);
218     GetCtrlVal(handle1,PANEL_MINS,&minutes);
219     GetCtrlVal(handle1,PANEL_CUM_VOLUME,&cum_volume);
220     Fmt(string,"%f[p2]%s%i%s%i%s%f%s",cum_volume," ml  ",hours," h ",minutes,
221          " min (",filter_rate,"ml/hr)\n\n");
222     SetCtrlVal(handle1,PANEL_HISTORY,string);
223
224
225  }
226     return(0);
227 }
228
229 /*****************************************************************************/
230
231 int change_filter_rate(int panel, int control, int event, void *callbackData,
232 int eventData1, int eventData2){
233 int return_code;
234 double new_filter_rate,filter_rate,blood_volume;
235 char string[100];
236
237 if(event==EVENT_COMMIT)
```

```
{

 /* switch system status lights to OFF*/

  SetCtrlVal(handle1,PANEL_RUNNING,0);
  SetCtrlVal(handle1,PANEL_STOPPED,1);


   /* get existing value of filter rate from screen as no longer doing this as a global */
   GetCtrlVal(handle1,PANEL_FILTER_RATE,&filter_rate);

  global_stop = 1;       /* this makes sure the run_system() function returns*/


   /* dim all controls while the filtration rate is being changed */

   SetCtrlAttribute (handle1,PANEL_INITIAL,ATTR_DIMMED,1);
   SetCtrlAttribute (handle1, PANEL_START, ATTR_DIMMED, 1);
   SetCtrlAttribute (handle1, PANEL_CHANGE_RATE, ATTR_DIMMED, 1);
   SetCtrlAttribute (handle1,PANEL_STOP,ATTR_DIMMED,1);
   SetCtrlAttribute (handle1,PANEL_QUIT,ATTR_DIMMED,1);


   ResetTextBox(handle1, PANEL_MESSAGE_BOX, "");
   InsertTextBoxLine(handle1,PANEL_MESSAGE_BOX,0," Enter the new ultrafiltration rate
                        that you require");
   InsertTextBoxLine(handle1,PANEL_MESSAGE_BOX,1," in the box below.");


/* stay in this loop until a new value for the ultrafiltration rate is entered */

/* set filter_rate to validate from indicator mode so a value can be entered */
 SetCtrlAttribute(handle1,PANEL_FILTER_RATE,ATTR_CTRL_MODE,VAL_VALIDATE);
do
 {
  SetActiveCtrl(handle1,PANEL_FILTER_RATE);
  GetUserEvent(1,&panel_event,&control_event);
  GetCtrlVal(handle1,PANEL_FILTER_RATE,&new_filter_rate);
 }
while(new_filter_rate == filter_rate);

 /* set filter_rate back to indicator */
 SetCtrlAttribute(handle1,PANEL_FILTER_RATE,ATTR_CTRL_MODE,VAL_VALIDATE);


/* get confirmation that new rate is to be used */

 ResetTextBox(handle1, PANEL_MESSAGE_BOX, "");
 Fmt(string,"%s%f[p2]%s","The ultrafiltration rate has been changed to ",
      new_filter_rate," ml/hr.\n");
 SetCtrlVal(handle1,PANEL_MESSAGE_BOX,string);
 InsertTextBoxLine(handle1,PANEL_MESSAGE_BOX,1,
 " Do you want to use this new rate?");

   return_code = ConfirmPopup("","Do you want to use the new filtration rate?");



/* act on responses from pop up panel */

   if(return_code == 1)
    {filter_rate = new_filter_rate;
     ResetTextBox(handle1, PANEL_MESSAGE_BOX, "");
```

```
301       InsertTextBoxLine(handle1,PANEL_MESSAGE_BOX,0," The filtration rate has been");
302       InsertTextBoxLine(handle1,PANEL_MESSAGE_BOX,1," changed to the new value.");
303       InsertTextBoxLine(handle1,PANEL_MESSAGE_BOX,2,
304                                 " You must now restart the system.");
305      }
306    if(return_code == 0)
307     {ResetTextBox(handle1, PANEL_MESSAGE_BOX, "");
308      SetCtrlVal(handle1,PANEL_FILTER_RATE,filter_rate);
309      InsertTextBoxLine(handle1,PANEL_MESSAGE_BOX,0," The filtration rate");
310      InsertTextBoxLine(handle1,PANEL_MESSAGE_BOX,1," is unchanged.");
311      InsertTextBoxLine(handle1,PANEL_MESSAGE_BOX,2,
312                                 " You must now restart the system.");
313     }
314
315     /* set controls back mostly to undimmed as you come out of this function */
316
317
318     SetCtrlAttribute (handle1,PANEL_INITIAL,ATTR_DIMMED,0);
319     SetCtrlAttribute (handle1, PANEL_START, ATTR_DIMMED, 0);
320     SetCtrlAttribute (handle1, PANEL_CHANGE_RATE, ATTR_DIMMED, 0);
321     SetCtrlAttribute (handle1,PANEL_STOP,ATTR_DIMMED,1);
322     SetCtrlAttribute (handle1,PANEL_QUIT,ATTR_DIMMED,0);
323
324     /* take focus away from filtration rate control */
325     SetActiveCtrl(handle1,PANEL_START);
326
327   }
328
329  return(0);
330  }
331
332
333  /*****************************************************************************/
334
335
336  int system_stop(int panel, int control, int event, void *callbackData,
337  int eventData1, int eventData2){
338   if(event==EVENT_COMMIT)
339     {
340
341     global_stop = 1;
342
343     ResetTextBox(handle1, PANEL_MESSAGE_BOX, "");
344
345     InsertTextBoxLine(handle1,PANEL_MESSAGE_BOX,0," The system is now stopped.");
346
347     SetCtrlVal(handle1,PANEL_RUNNING,0);
348     SetCtrlVal(handle1,PANEL_STOPPED,1);
349
350     SetCtrlAttribute (handle1,PANEL_INITIAL,ATTR_DIMMED,0);
351     SetCtrlAttribute (handle1, PANEL_START, ATTR_DIMMED, 0);
352     SetCtrlAttribute (handle1, PANEL_CHANGE_RATE, ATTR_DIMMED, 0);
353     SetCtrlAttribute (handle1,PANEL_STOP,ATTR_DIMMED,1);
354     SetCtrlAttribute (handle1,PANEL_QUIT,ATTR_DIMMED,0);
355
356
357
358     }
359  return(0);
360 }
361
362
363  /*****************************************************************************/
```

```
364
365 int quit_system(int panel, int control, int event, void *callbackData,
366                 int eventData1, int eventData2){
367 if(event==EVENT_COMMIT)
368   QuitUserInterface(0);
369 return(0);
370 }
371
```

INITIALISE.C

```
1 #include <header.h>
2
3
4                                   /* INITIALISE.C */
5
6                       /* FUNCTIONS IN THIS FILE:
7
8                                     INIT_HARDWARE()
9                                     TEST_HARDWARE()
10                                    TEST_STEPPER_MOTOR()
11                                    TEST_TRANSDUCER_AND_LINE()
12                                    INIT_SOFTWARE()
13
14
15
16
17 /*****************************************************************************/
18
19 int init_hardware()
20
21 {
22  int return_code;
23
24
25 ResetTextBox (handle1, PANEL_MESSAGE_BOX, "");
26 InsertTextBoxLine(handle1,PANEL_MESSAGE_BOX,0,"If the blood circuit is
27                       attached to the machine, remove it now.");
28
29 ProcessSystemEvents();
30 Delay(2);
31
32 MessagePopup("","Confirm that the blood circuit is not attached by pressing OK.");
33
34 ResetTextBox (handle1, PANEL_MESSAGE_BOX, "");
35 InsertTextBoxLine(handle1,PANEL_MESSAGE_BOX,0,"If the machine is switched off,
36 switch it  on now.");
37 ProcessSystemEvents();
38 Delay(2);
39 MessagePopup("","Confirm that the machine is switched on by pressing OK.");
40
41
42 ResetTextBox (handle1, PANEL_MESSAGE_BOX, "");
43 InsertTextBoxLine(handle1,PANEL_MESSAGE_BOX,0,"The system is now being checked...");
44 ProcessSystemEvents();
45
46 return_code = test_hardware(CIRCUIT_OUT);
47 if(return_code != 0)
48   return(return_code);
49
50 InsertTextBoxLine(handle1,PANEL_MESSAGE_BOX,1,"The system is working properly.");
```

```
51 ProcessSystemEvents();
52 Delay(2);
53
54 InsertTextBoxLine(handle1,PANEL_MESSAGE_BOX,2,
55 "Now attach the blood circuit to the machine.");
56 InsertTextBoxLine(handle1,PANEL_MESSAGE_BOX,3,
57 "Do not forget to plug in the pressure transducer.");
58 ProcessSystemEvents();
59 Delay(2);
60
61 MessagePopup("","Confirm that the blood circuit is in place by pressing OK.");
62 /* this repeats the above tests except for steppers */
63
64 ResetTextBox (handle1, PANEL_MESSAGE_BOX, "");
65 InsertTextBoxLine(handle1,PANEL_MESSAGE_BOX,0,
66 "The system is now being checked again....");
67 ProcessSystemEvents();
68 return_code = test_hardware(CIRCUIT_IN);
69 if(return_code != 0)
70   return(return_code);
71
72
73 /* finally, test pressure transducer */
74
75
76 return_code = test_transducer_and_line();
77 if(return_code != 0)
78   return(return_code);
79
80 InsertTextBoxLine(handle1,PANEL_MESSAGE_BOX,1,"The system is working properly.");
81 ProcessSystemEvents();
82 Delay(4);
83 /* normal return */
84
85 return(0);
86 }
87
88 /*****************************************************************************/
89
90
91 int test_hardware(short circuit_status)
92
93 {
94 short error_status,return_code;
95 double v_ref;
96
97 /*  TEST SERVOS */
98
99 /* servo A */
100
101 return_code = MoveServo(1,1);
102 if(return_code != 0)
103   return(return_code);
104
105
106
107 return_code = MoveServo(1,2);
108 if(return_code != 0)
109   return(return_code);
110
111
112
113 return_code = MoveServo(1,1);
```

```
114 if(return_code != 0)
115   return(return_code);
116
117 /* servo B */
118
119 return_code = MoveServo(2,1);
120 if(return_code != 0)
121   return(return_code);
122
123
124
125 return_code = MoveServo(2,2);
126 if(return_code != 0)
127   return(return_code);
128
129
130
131return_code = MoveServo(2,1);
132 if(return_code != 0)
133   return(return_code);
134
135
136
137
138
139 /* now test stepper motors if blood circuit is not attached */
140
141 if(circuit_status == CIRCUIT_OUT)
142 {
143
144  /* open taps first - this is to prevent pressure buildup if circuit has
145     been left in accidently*/
146   return_code = MoveServo(1,2);
147  if(return_code != 0)
148   return(return_code);
149
150   return_code = MoveServo(2,2);
151  if(return_code != 0)
152   return(return_code);
153
154
155   return_code = test_stepper_motor(1);
156   if(return_code != 0)
157    return(return_code);
158
159   return_code = test_stepper_motor(2);
160   if(return_code != 0)
161    return(return_code);
162
163
164  /* close taps again */
165
166
167      return_code = MoveServo(1,1);
168  if(return_code != 0)
169   return(return_code);
170
171   return_code = MoveServo(2,1);
172  if(return_code != 0)
173   return(return_code);
174
175
176
```

```
177
178 }
179
180
181 /* test pressure transducer */
182
183
184
185 error_status = AI_VRead(1,0,2,&v_ref);
186 /*  if(circuit_status == CIRCUIT_OUT)
187    if(v_ref > UNPLUGGED_LIMIT )
188       return(17);                    */
189  if(circuit_status == CIRCUIT_IN)
190    if(v_ref > AMB_HI_LIMIT || v_ref < AMB_LO_LIMIT )
191       return(18);
192
193 return(0);
194 }
195
196
197
198
199 /*****************************************************************************/
200
201 int test_stepper_motor(short motor_num)
202
203 {
204 short switch_line,switch_state,dirn_line,clock_line;
205 int step_counter = 0;
206 double mark;
207
208 /* test stepper motor */
209
210 if(motor_num == 1)
211
212  {switch_line = SWITCH_A;
213   clock_line = CLOCK_A;
214   dirn_line = DIRN_A;
215  }
216
217else
218
219 {switch_line = SWITCH_B;
220  clock_line = CLOCK_B;
221  dirn_line = DIRN_B;
222 }
223
224  mark = Timer();
225  DIG_Out_Line (1, 0, dirn_line, DOWN);
226
227   .
228 /*move plunger to bottom of syringe*/
229
230 DIG_In_Line(1, 1, switch_line, &switch_state); /*read plunger switch*/
231
232 while(switch_state == 0)
233
234 {
235  SyncWait(mark,0.025 * step_counter);
236
237  DIG_Out_Line(1,0,clock_line,0);
238  Delay(.0001);
239  DIG_Out_Line(1,0,clock_line,1);
```

```
240  step_counter++;
241  DIG_In_Line(1, 1, switch_line, &switch_state); /*read plunger switch*/
242
243   /* this bit checks that the plunger hasn't gone too far - i.e. checks that
244     stepper is working and also the microswitch */
245
246       if(step_counter > NO_OF_STEPS_LIMIT)
247       return(15);
248
249 }
250
251 /* Now move up a few steps to check microswitch hasn't failed to 1 state */
252
253 mark = Timer();
254 DIG_Out_Line (1, 0, dirn_line, UP);
255 step_counter = 0;
256
257
258 while(step_counter < 25)
259
260  {SyncWait(mark,0.025 * step_counter);
261
262   DIG_Out_Line(1,0,clock_line,0);
263   Delay(.0001);
264   DIG_Out_Line(1,0,clock_line,1);
265   step_counter++;
266  }
267
268 DIG_In_Line(1, 1, switch_line, &switch_state); /*read plunger switch*/
269 if(switch_state == 1)
270   return(16);
271
272 /* Now move the plunger back down again-these 2 segments could be made into a function */
273 mark = Timer();
274 DIG_Out_Line (1, 0, dirn_line, DOWN);
275 step_counter = 0;
276
277 DIG_In_Line(1, 1, switch_line, &switch_state); /*read plunger switch*/
278
279 while(switch_state == 0)
280
281 {
282  SyncWait(mark,0.025 * step_counter);
283
284  DIG_Out_Line(1,0,clock_line,0);
285  Delay(.0001);
286  DIG_Out_Line(1,0,clock_line,1);
287  step_counter++;
288  DIG_In_Line(1, 1, switch_line, &switch_state); /*read plunger switch*/
289
290   /* this bit checks that the plunger hasn't gone too far - i.e. checks that
291     stepper is working and also the microswitch */
292
293  if(step_counter > NO_OF_STEPS_LIMIT)
294       return(15);
295 }
296
297 return(0);
298 }
299
300 /*****************************************************************************/
301
302 /* this function withdraws 1ml of blood to test line and then puts it back */
```

```
303
304 int test_transducer_and_line()
305
306 {
307  short error_status;
308  int return_code,step_counter;
309  double mark,v_ref,v_upper,v_lower;
310
311
312  /* move taps to A closed and B open */
313
314  return_code = MoveServo(1,1);
315  if(return_code != 0)
316   return(return_code);
317
318  return_code = MoveServo(2,2);
319  if(return_code != 0)
320   return(return_code);
321
322  /* withdraw 1ml of blood */
323
324  return_code = Withdraw_Blood(1.0);
325  if(return_code != 0)
326   return(return_code);
327
328  Delay(2);    /* not essential */
329
330  /* close tap B */
331
332  return_code = MoveServo(2,1);
333  if(return_code != 0)
334   return(return_code);
335
336  /* read voltage for reference */
337
338 error_status = AI_VRead(1,0,2,&v_ref);
339
340  /* move plunger B down */
341
342 mark = Timer();
343 DIG_Out_Line (1, 0, DIRN_B, DOWN);
344 step_counter = 0;
345
346 while(step_counter < 20)
347
348  {SyncWait(mark,0.025 * step_counter);
349
350   DIG_Out_Line(1,0,CLOCK_B,0);
351   Delay(.0001);
352   DIG_Out_Line(1,0,CLOCK_B,1);
353   step_counter++;
354  }
355
356 error_status = AI_VRead(1,0,2,&v_upper);
357
358 if(v_upper - v_ref < ONEML_P_CHNGE)
359   return(19);
360
361 Delay(2);    /* not essential */
362
363  /* now move plunger B up */
364
365 mark = Timer();
```

```
366 DIG_Out_Line (1, 0, DIRN_B, UP);
367 step_counter = 0;
368
369 while(step_counter < 40)
370
371  {SyncWait(mark,0.025 * step_counter);
372
373   DIG_Out_Line(1,0,CLOCK_B,0);
374   Delay(.0001);
375   DIG_Out_Line(1,0,CLOCK_B,1);
376   step_counter++;
377  }
378
379 error_status = AI_VRead(1,0,2,&v_lower);
380
381 if(v_ref - v_lower < ONEML_P_CHNGE)
382   return(20);
383
384 Delay(2);
385
386 /* now move plunger back to neutral position */
387
388 mark = Timer();
389 DIG_Out_Line (1, 0, DIRN_B, DOWN);
390 step_counter = 0;
391
392 while(step_counter < 20)
393
394  {SyncWait(mark,0.025 * step_counter);
395
396   DIG_Out_Line(1,0,CLOCK_B,0);
397   Delay(.0001);
398   DIG_Out_Line(1,0,CLOCK_B,1);
399   step_counter++;
400  }
401
402
403 /* open tap B */
404
405 return_code = MoveServo(2,2);
406  if(return_code != 0)
407   return(return_code);
408
409
410 /* return blood */
411
412 return_code = Return_Blood();
413  if(return_code != 0)
414    return(return_code);
415
416 /*.return taps to original position - i.e. tap A open and tap B closed */
417
418 return_code = MoveServo(1,2);
419  if(return_code != 0)
420   return(return_code);
421
422 return_code = MoveServo(2,1);
423  if(return_code != 0)
424   return(return_code);
425
426
427 return(0);
428
```

```
429 }
430
431
432 /*************************************************************************/
433
434
435 int init_software()
436
437 {
438  int box_status,return_code;
439  double birth_weight,blood_volume,filter_rate;
440
441
442  ResetTextBox (handle1, PANEL_MESSAGE_BOX, "");
443  box_status = InsertTextBoxLine (handle1, PANEL_MESSAGE_BOX, 0,
444  " Enter the weight of the baby (in grammes).");
445
446  /* set birth weight to validate from indicator mode so a value can be entered */
447  SetCtrlAttribute(handle1,PANEL_BIRTH_WEIGHT,ATTR_CTRL_MODE,VAL_VALIDATE);
448
449  do
450  {
451   SetActiveCtrl(handle1,PANEL_BIRTH_WEIGHT);
452   GetUserEvent(1,&panel_event,&control_event);
453   GetCtrlVal(handle1,PANEL_BIRTH_WEIGHT,&birth_weight);
454  }
455  while(birth_weight == 0.0);
456
457  /* put birth weight back to indicator so it can't be changed */
458  SetCtrlAttribute(handle1,PANEL_BIRTH_WEIGHT,ATTR_CTRL_MODE,VAL_INDICATOR);
459
460  blood_volume = (85.0*birth_weight/1000)*0.06;
461  SetCtrlVal(handle1,PANEL_WB_VOLUME,blood_volume);
462
463  box_status = DeleteTextBoxLine (handle1, PANEL_MESSAGE_BOX,0);
464 InsertTextBoxLine(handle1,PANEL_MESSAGE_BOX,0,
465 "The working blood volume shown below has been");
466  InsertTextBoxLine(handle1,PANEL_MESSAGE_BOX,1,"calculated from the body weight");
467  InsertTextBoxLine(handle1,PANEL_MESSAGE_BOX,2,"you entered(6% of total blood vol.");
468  InsertTextBoxLine(handle1,PANEL_MESSAGE_BOX,3,", assuming a volume of 85ml/kg.)");
469  InsertTextBoxLine(handle1,PANEL_MESSAGE_BOX,4,
470  "Do you want to use this calculated figure?");
471
472    return_code = ConfirmPopup("","Do you want to use the calculated figure?");
473
474
475
476    if(return_code == 1)
477      ResetTextBox (handle1, PANEL_MESSAGE_BOX, "");
478    if(return_code == 0)
479     {ResetTextBox (handle1, PANEL_MESSAGE_BOX, "");
480 InsertTextBoxLine(handle1,PANEL_MESSAGE_BOX,0,
481 "Enter your own value for the working");
482  InsertTextBoxLine(handle1,PANEL_MESSAGE_BOX,1," blood volume in the box below.");
482
483      /* set wb_volume to validate so a value can be entered */
484      SetCtrlAttribute(handle1,PANEL_WB_VOLUME,ATTR_CTRL_MODE,VAL_VALIDATE);
485
486      do
487        {SetActiveCtrl(handle1,PANEL_WB_VOLUME);
488         GetUserEvent(1,&panel_event,&control_event);
489         GetCtrlVal(handle1,PANEL_WB_VOLUME,&blood_volume);
490        }
```

```
491      while(control_event != PANEL_WB_VOLUME);
492
493      /* set wb_volume back to indicator */
494      SetCtrlAttribute(handle1,PANEL_WB_VOLUME,ATTR_CTRL_MODE,VAL_INDICATOR);
495
496      ResetTextBox(handle1, PANEL_MESSAGE_BOX, "");
497     }
498
499
500 InsertTextBoxLine(handle1,PANEL_MESSAGE_BOX,0," Now enter the ultrafiltration rate");
501 InsertTextBoxLine(handle1,PANEL_MESSAGE_BOX,1," that you require (in ml/hour).");
502
503
504     /* set filter_rate to validate so a value can be entered */
505     SetCtrlAttribute(handle1,PANEL_FILTER_RATE,ATTR_CTRL_MODE,VAL_VALIDATE);
506     do
507     {
508      SetActiveCtrl(handle1,PANEL_FILTER_RATE);
509      GetUserEvent(1,&panel_event,&control_event);
510      GetCtrlVal(handle1,PANEL_FILTER_RATE,&filter_rate);
511     }
512       while(filter_rate == 0.0);
513
514     /* set filter_rate back to indicator */
515     SetCtrlAttribute(handle1,PANEL_FILTER_RATE,ATTR_CTRL_MODE,VAL_INDICATOR);
516
517
518
519  return(0);
520 }
```

ERROR.C

```
1
2
3 /********************** ERROR.C ******************************/
4
5
6 /* FUNCTION LIST:
7
8       process_error() */
9
10
11
12 #include <Header.h>
13
14
15
16
17
18
19 int process_error(int error_code)
20
21
22 {
23  /* switch system status lights to OFF*/
24
25   SetCtrlVal(handle1,PANEL_RUNNING,0);
26   SetCtrlVal(handle1,PANEL_STOPPED,1);
27
28      if(software_init == TRUE)
```

```
29
30    {SetCtrlAttribute (handle1,PANEL_INITIAL,ATTR_DIMMED,0);
31     SetCtrlAttribute (handle1, PANEL_START, ATTR_DIMMED, 0);
32     SetCtrlAttribute (handle1, PANEL_CHANGE_RATE, ATTR_DIMMED, 0);
33
34     SetCtrlAttribute (handle1,PANEL_STOP,ATTR_DIMMED,1);
35     SetCtrlAttribute (handle1,PANEL_QUIT,ATTR_DIMMED,0);
36       }
37
38     if(software_init == FALSE)
39
40    {SetCtrlAttribute (handle1,PANEL_INITIAL,ATTR_DIMMED,0);
41     SetCtrlAttribute (handle1, PANEL_START, ATTR_DIMMED, 1);
42     SetCtrlAttribute (handle1, PANEL_CHANGE_RATE, ATTR_DIMMED,
43     1);
44     SetCtrlAttribute (handle1,PANEL_STOP,ATTR_DIMMED,1);
45     SetCtrlAttribute (handle1,PANEL_QUIT,ATTR_DIMMED,0);
46       }
47
48
49
50   ResetTextBox(handle1,PANEL_MESSAGE_BOX,"");
51
52
53  switch(error_code)
54    {
55        case 1:
56  InsertTextBoxLine(handle1,PANEL_MESSAGE_BOX,0,"Error code 1: tap
57  drivers could not find required position.");
58              break;
59
60        case 2:
61
62 InsertTextBoxLine(handle1,PANEL_MESSAGE_BOX,0,"Error code 2:
63 blood withdrawal has taken longer than 120 seconds - check venous line
64 for blockages.");
65                        break;
66
67        case 3:
68
69 InsertTextBoxLine(handle1,PANEL_MESSAGE_BOX,0,"Error code 3:
70 Servo detector failure to zero state.");
71                        break;
72
73        case 4:
74
75 InsertTextBoxLine(handle1,PANEL_MESSAGE_BOX,0,"Error code 4:
76 Failure in Return_To_Start_Posn().");
77                        break;
78
79        case 5:
80
81 InsertTextBoxLine(handle1,PANEL_MESSAGE_BOX,0,"Error code 5:
82 Step limit exceeded in return_blood().");
83                        break;
84
85        case 6:
86
87 InsertTextBoxLine(handle1,PANEL_MESSAGE_BOX,0,"Error code 6:
88 Error in Withdraw_Blood(). Plunger not at bottom");
89
90 InsertTextBoxLine(handle1,PANEL_MESSAGE_BOX,1,"of syringe or
91 microswitch failed to zero state.");
```

```
92                      break;
93
94      case 7:
95
96 InsertTextBoxLine(handle1,PANEL_MESSAGE_BOX,0,"Error code 7:
97 Error in Withdraw_Blood(). Suspect failure switch B");
98
99  InsertTextBoxLine(handle1,PANEL_MESSAGE_BOX,1,"to 1 state or loss
100 of drive to stepper motors.");
101                     break;
102
103     case 8:
104
105 InsertTextBoxLine(handle1,PANEL_MESSAGE_BOX,0,"Error code 8:
106 Error in Filter_Blood(). Suspect failure push plunger");
107
108 InsertTextBoxLine(handle1,PANEL_MESSAGE_BOX,1,"microswitch to 0
109 state or loss of drive to push stepper motor.");
110                     break;
111     case 9:
112
113 InsertTextBoxLine(handle1,PANEL_MESSAGE_BOX,0,"Error code 9:
114 Error in Filter_Blood(). Suspect failure in");
115
116 InsertTextBoxLine(handle1,PANEL_MESSAGE_BOX,1,"pull microswitch
117 to 1 state.");
118                     break;
119
120     case 10:
121
122 InsertTextBoxLine(handle1,PANEL_MESSAGE_BOX,0,"Error code 10:
123 Pressure limits exceeded in Filter_Blood().");
124                     break;
125
126     case 11:
127
128 InsertTextBoxLine(handle1,PANEL_MESSAGE_BOX,0,"Error code 11:
129 Pressure limits exceeded in Return_To_Start_Posn().");
130                     break;
131
132
133     case 12:
134
135 InsertTextBoxLine(handle1,PANEL_MESSAGE_BOX,0,"Error code 12:
136 Pressure limits exceeded in Return_Blood().");
137                     break;
138
139
140     case 13:
141
142 InsertTextBoxLine(handle1,PANEL_MESSAGE_BOX,0,"Error code 13:
143 Pressure limits exceeded in Withdraw_Blood().");
144                     break;
145
146
147     case 14:
148
149 InsertTextBoxLine(handle1,PANEL_MESSAGE_BOX,0,"Error code 14:
150 Pressure limits exceeded in Reverse_Flow().");
151                     break;
152
153     case 15:
154
```

```
155 InsertTextBoxLine(handle1,PANEL_MESSAGE_BOX,0,"Error code 15:
156 Stepper or microswitch failure in init_machine().");
157                         break;
158
159       case 16:
160
161 InsertTextBoxLine(handle1,PANEL_MESSAGE_BOX,0,"Error code 16:
162 Microswitch failure to 1 state in init_machine().");
163                         break;
164
165       case 17:
166
167 InsertTextBoxLine(handle1,PANEL_MESSAGE_BOX,0,"Error code 17:
168 Pressure amplifier circuit failure in init_machine().");
169                         break;
170
171       case 18:
172
173 InsertTextBoxLine(handle1,PANEL_MESSAGE_BOX,0,"Code 18:
174 Pressure transducer/amplifier failure in init_machine().");
175                         break;
176
177       case 19:
178
179 InsertTextBoxLine(handle1,PANEL_MESSAGE_BOX,0,"Code 19:Error in
180 init_machine.Transducer not responding to pressure rise.");
181                         break;
182
183       case 20:
184
185 InsertTextBoxLine(handle1,PANEL_MESSAGE_BOX,0,"Code 20:Error in
186 init_machine(). Transducer not responding to pressure drop");
187                         break;
188
189       case 21:
190
191 InsertTextBoxLine(handle1,PANEL_MESSAGE_BOX,0,"Code 21:
192 Computer fault: loss of timing control - check computer setup. ");
193                         break;
194
195     }
196
197  /* this message is the same for all error codes */
198
199
200   InsertTextBoxLine(handle1,PANEL_MESSAGE_BOX,1,"The system
201  has been stopped.");
202
203  /* put this back in later */
204
205
206  /* use a child panel as a pop up to reset the alarm */
207
208   DisplayPanel(handle3);
209
210    do
211      {GetUserEvent(0,&panel_event,&control_event);
212       Beep();
213      }
214     while(!(panel_event == handle3 && control_event == ALARM_OK));
215
216     Delay(0.5);
217     HidePanel(handle3);
```

```
218
219
220 return(0);
221 }
```

UPDTIMER.C

```
1  /* UpdTimer.c  -- this function takes the output from Timer(), and converts
2   it into hours and minutes, and then displays it.*/
3
4
5  /* FUNCTION LIST:
6
7                          Update_Timer()
8                          Reset_Timer()
9
10 */
11
12
13 #include <Header.h>
14
15
16
17 int Update_Timer(double start_time)
18
19 {
20
21  double tot_seconds,seconds;
22  int    minutes,hours;
23
24
25 tot_seconds = Timer() - start_time;
26
27 hours = floor(tot_seconds/3600);
28 tot_seconds = tot_seconds - (hours * 3600);
29 minutes = floor(tot_seconds/60);
30 seconds = tot_seconds -(minutes * 60);
31
32
33 SetCtrlVal(handle1,PANEL_HOURS,hours);
34 SetCtrlVal(handle1,PANEL_MINS,minutes);
35
36
37
38 return(0);
39 }
40
41
42 /*****************************************************************/
43
44 /*this function simply resets the screen 'total treatment time' to zero each time the system
45   is restarted */
46
47 double Reset_Timer()
48
49  { SetCtrlVal(handle1,PANEL_HOURS,0);
50    SetCtrlVal(handle1,PANEL_MINS,0);
51    return(Timer());
52  }
```

ROUND.C

```
1                /*ROUND.C*/
2
3
4       /*FUNCTION LIST:
5
6                Round()
7                                        */
8
9   #include <header.h>
10
11
12  /*************************************************************/
13
14      int Round(double number)
15
16   {
17        if (ceil(number) - number > number - floor(number))
18
19                return(floor(number));
20
21        else
22
23           {if((ceil(number) - number == number - floor(number))
24                && number < 0.0)
25
26               return(floor(number));
27
28            else return(ceil(number));
29                }
30
31      }
32
33  /*************************************************************/
```

HAEMO1.H

```
/*****************************************************************************/
/* LabWindows/CVI User Interface Resource (UIR) Include File                 */
/* Copyright (c) National Instruments 1999. All Rights Reserved.             */
/*                                                                           */
/* WARNING: Do not add to, delete from, or otherwise modify the contents     */
/*          of this include file.                                            */
/*****************************************************************************/

#include <userint.h>

#ifdef __cplusplus
    extern "C" {
#endif

     /* Panels and Controls: */

#define  ALARM                   1
#define  ALARM_OK                2
#define  ALARM_TEXTMSG           3

#define  PANEL                   2
#define  PANEL_INITIAL           2       /* callback function: system_init */
#define  PANEL_START             3       /* callback function: system_start */
```

```
#define  PANEL_STOP            4      /* callback function: system_stop */
#define  PANEL_BIRTH_WEIGHT           5
#define  PANEL_WB_VOLUME              6
#define  PANEL_HOURS                  7
#define  PANEL_FILTER_RATE            8
#define  PANEL_CHANGE_RATE            9      /* callback function: change_filter_rate */
#define  PANEL_CUM_VOLUME            10
#define  PANEL_MESSAGE_BOX           11
#define  PANEL_MINS                 12
#define  PANEL_QUIT                 13     /* callback function: quit_system */
#define  PANEL_HISTORY               14
#define  PANEL_STOPPED               15
#define  PANEL_RUNNING               16
#define  PANEL_ACCESS_MODE           17
#define  PANEL_GRAMMES               18
#define  PANEL_MILLILITRES           19
#define  PANEL_MILLILITRES_HOUR          20
#define  PANEL_DECORATION                21
#define  PANEL_DECORATION_2              22
#define  PANEL_TEXTMSG                   23
#define  PANEL_TEXTMSG_3                 24
#define  PANEL_TEXTMSG_4                 25
#define  PANEL_TEXTMSG_2                 26

#define  RESPONSE                    3
#define  RESPONSE_YES                2
#define  RESPONSE_NO                 3


    /* Menu Bars, Menus, and Menu Items: */

        /* (no menu bars in the resource file) */


    /* Callback Prototypes: */

int  CVICALLBACK change_filter_rate(int panel, int control, int event, void *callbackData, int
eventData1, int eventData2);
int  CVICALLBACK quit_system(int panel, int control, int event, void *callbackData, int eventData1,
int eventData2);
int  CVICALLBACK system_init(int panel, int control, int event, void *callbackData, int eventData1,
int eventData2);
int  CVICALLBACK system_start(int panel, int control, int event, void *callbackData, int
eventData1, int eventData2);
int  CVICALLBACK system_stop(int panel, int control, int event, void *callbackData, int
eventData1, int eventData2);


#ifdef __cplusplus
    }
#endif
```

HEADER.H

```
1 /* HEADER.H */
2
3 #include <utility.h>
4 #include <dataacq.h>
5 #include <ansi_c.h>
6 #include <easyio.h>
7 #include <userint.h>
```

```
8 #include <haemo1.h>
9 #include <formatio.h>
10
11 /* these are the defines for the pump4.c functions */
12
13 #define ANTI 0
14 #define CLOCK 1
15 #define DIRN_A 0
16 #define DIRN_B 1
17 #define CLOCK_A 2
18 #define CLOCK_B 3
19 #define UP     1
20 #define DOWN   0
21 #define SWITCH_A 4
22 #define SWITCH_B 5
23
24 #define TRUE 1
25 #define FALSE 0
26
27 #define CIRCUIT_IN 1                    /*these are used in initialise.c */
28 #define CIRCUIT_OUT 0
29
30
31 /* these are the defines for the servo functions */
32
33 #define ANTI_1   5                              /* these define the output drive lines*/
34 #define CLOCK_1  4                                      /* to the motors */
35 #define ANTI_2   7
36 #define CLOCK_2  6
37
38 #define A1   0                                          /* these define the sensor input lines */
39 #define A2   1
40 #define B1   2
41 #define B2   3
42
43
44 #define NONE  -1  /*this is assigned to be out of range for the output port*/
45
46
47
48
49 int run_system(double blood_volume,double f_rate);
50 int Filter_Blood(double f_rate,double* volume_filtered);
51 int Return_To_Start_Posn(void);
52 int Withdraw_Blood(double blood_volume);
53 int Return_Blood(void);
54 int Reverse_Flow(int *step_counter,double v_ref);
55 int Round(double number);      /*in file round.c*/
56 int Update_Timer(double);
57 double Reset_Timer(void);
58 int process_error(int error_code);
59 int init_hardware(void);        /* in file initialise.c */
60 int test_stepper_motor(short motor_num);
61 int test_hardware(short circuit_status);
62 int test_transducer_and_line(void);
63 int init_software(void);
64
65 /*these functions are in the file servo2.c*/
66 int MoveServo(int Servo_Num,int Target_Posn);
67 int Current_Posn(int i_line1,int i_line2);
68 int Move_To_Home_Posn(int i_line1,int i_line2,int output_line);
69 int Change_Direction(short *o_line);
70 int Check_Servo_Posn(int i_line1,int i_line2,short output_line);
```

```
71
72
73
74 int panel_event,control_event;
75 int handle1,handle2,handle3;
76 int global_stop; /* only using this because can't get the return code from
77                     get user event to work */
78
79 int hardware_init;
80 int software_init;
81
82
83 /* MACHINE PARAMETERS */
84
85 #define NUM_STEPS_PER_ML  246    /* this depends on the diameter of the syringe */
86 #define STEP_RATE 40 /*units are steps/sec. - this is the underlying rate for blood pumping*/
87 #define NO_OF_STEPS_LIMIT 2500 /* this is the max no. of steps */
88                                  /* allowed in a single stroke of the syringe */
89 #define VOL_TIME_FACTOR 30       /* this is the conversion factor used in Withdraw_Blood */
90
91
92 #define FILT_CORRECTION_FACTOR 1.104  /* this is the correction factor used to
93               compensate for the differences between blood and water filtration rates */
94
95
96
97
98 /*#define DEF_LP_LIMIT 0.15*/ /* these define upper and lower limits for safety monitoring */
99 /*#define DEF_HP_LIMIT 2.0 */ /* changed from 1.95 they are based on a gain of 5 */
100                                /* which gives 0 - 2v range*/
101 /* this gain might need changing later */
102
103 /* the above limits could be used for withdrawal and return etc
104   below tighter limits are defined for use in the Filter_Blood function, as
105   the pressures in this function should not vary very much at all */
106
107 /*#define FILTER_LP_LIMIT 0.5    */  /*changed from 1.3*/
108 /*#define FILTER_HP_LIMIT 2.0    */  /*changed from 1.7*/
109
110
111 /* changed all the above 11/9/98 to new system - values below */
112
113 #define UNPLUGGED_LIMIT 0.8      /* this is the alarm limit for the diff.amp with the */
114                                  /* transducer unplugged -normal value should be ~0.62 v */
115 #define AMB_HI_LIMIT 3.2         /* these limits are for the analogue readout at */
116 #define AMB_LO_LIMIT 2.8         /* ambient pressure - the pot should be adjusted to */
117                                  /* give an ambient readout of 3.0 v */
118 #define ONEML_P_CHNGE 0.2        /* this is the min. pressure change required during the
119                                     transducer and line test in initialise.c */
120 #define FILT_P_RISE  2.0                 /* changed from 1.5 */
121 #define FILT_P_DROP  3.5                 /*changed from 1.0*/
122 #define WITHRET_P_RISE  2.0              /* changed from 1.0*/
123 #define WITHRET_P_DROP  1.5
124 #define WITH_DELAY_LIMIT 1.3     /* a pressure drop greater than 1.3v will produce a delay
125                                  /* in blood withdrawal */
```

APPENDIX E Operating Instructions for Haemodialysis System

SETTING UP THE MACHINE

1. If this has not already been done, you must wire up the computer, video monitor etc. Refer to figure E.1 for a diagram of how everything connects together. It is **VERY IMPORTANT** that the monitor, computer and dialyser are all connected to the mains through the isolation transformer. **NEVER** connect any of these direct to the mains supply.

Connect everything together as shown in the diagram. The video cable can only connect to one socket on the back of the computer, so it is not possible to plug it into the wrong place. The keyboard and mouse plug into sockets near the top of the back of the computer. These sockets have symbols next to them to tell you which is for the keyboard and which for the mouse.

2. Connect the ribbon cable between the machine and the computer. There is only one socket that the cable can connect to on the computer, and the shape of the plugs prevents them being connected the wrong way round. Connect the grey RCD mains lead into the back of the dialyser.

3. Switch on the isolation transformer and then the computer. Press in the grey button on the top of the grey RCD plug that supplies the dialyser. Check that red bars appear in the window on the back of the plug. **Do not switch on the machine at this stage.**

4. Once Windows 95 has loaded, you will see 3 icons on the right hand side of the desktop. The first is the normal dialysis system. The second is the 'fast' dialysis system, with increased blood flow rates. The differences between these two will be explained later. The third icon is a manual control system, which will also be explained later. Choose whichever of the two main systems you want to use, and double click on its icon with the mouse.

5. The user interface for the system will appear on the screen. The various components of this interface will be explained later. For now, the only button you need is the 'Initialise System' button at the top left of the screen.

6. Press the 'Initialise System' button. This will take you through a testing and setting up procedure. Follow all the instructions given in the message box at the top centre of the screen. Respond to questions by clicking on the Yes, No and OK buttons (with the mouse) that will appear during the procedure.

7. Connect the blood circuit (see figure E.2 for how to construct the circuit) to the machine when prompted by the user interface. Slot both 3 way taps into their recesses and push the syringes into their holders. The 3 way taps are quite difficult to fit into their recesses so be patient. If they don't appear to line up make sure that the right hand syringe is screwed fairly tight into the 3 way tap and that all other luer locks are tightly fitted together. Once the taps are in place, push the clamps fully over them and then tighten down the retaining screws. Do the same for the syringe clamps - i.e. push the clamps fully down before tightening the retaining screws. Don't forget to plug the pressure transducer in.

8. If the initialisation procedure fails for any reason, quit the user interface and run it again. You don't need to take the blood circuit out if you have already attached it - but you must disconnect the pressure transducer before restarting the initialisation procedure. Plug it back in when the user interface prompts you to attach the blood circuit.

9. If the machine has already been set up the initialisation procedure can be bypassed, but check that the correct parameters are entered in each box on the interface, particularly the working blood volume and the ultrafiltration rate.

10. Once you have connected the blood circuit to the machine, **make sure the venous line is open and attached to a source of normal saline.** This is important because the system needs to withdraw a small amount of fluid (approx. 1 ml) through the venous line for testing purposes.

11. When the procedure reaches the priming stage, the circuit can be primed as follows: attach a 10 ml syringe to the tap at the bottom of the left hand syringe (tap 1 in figure 2). You may have to add another 3 way tap to tap 1 so that the 10 ml syringe does not get in the way of the syringe driver mechanism. Remove the saline bag from the venous line and attach the venous line to the patient. Turn tap 1 so that the syringe is closed off but the filter and the 10 ml syringe are both open. The filter can now be primed by withdrawing the syringe barrel so that blood flows from the patient into the circuit. Stop when the blood reaches tap 1. **Do not forget to turn tap 1 back so that the syringe and the haemofilter are both open and the left hand port is closed.**

12. Once the filter has been primed, the system is ready for use. You will be prompted to press the 'Start Treatment' button to start the machine.

FINISHING TREATMENT

When the treatment session is finished, the blood remaining in the filter can be flushed back into the patient using the following procedure:
Press the 'Stop Treatment' button when the machine comes to the end of returning blood to the patient. If treatment is stopped at this point then both syringes will be empty of blood. Quit from the user interface and run the manual control system by double clicking on the icon on the desktop. Open both taps. You can then flush the remaining blood back into the patient using a saline filled syringe attached to tap 1.

DESCRIPTION OF THE USER INTERFACE

Refer to figure E.3.
The buttons at the top left are the main controls.

Initialise System: starts the initialisation procedure.

Start/Restart Treatment: starts the machine once the initialisation procedure has been completed. Restarts the machine if it has been stopped or the ultrafiltration rate has been changed.

Change Filtration Rate: if you want to change the ultrafiltration rate at any point, whether the machine is running or not, press this button. Follow the on screen instructions to change the filtration rate. Once this has been done you will be prompted to restart the machine by pressing the 'Start/Restart Button'.

Stop Treatment: stops the machine. In an emergency the machine can always be stopped by switching off the mains power to it.

Quit: this quits out of the user interface and returns the computer to Windows 95.

The white box at top centre displays system messages and gives instructions to the user.
The white box at top right displays the treatment history. For each period of treatment at a given ultrafiltration rate, the following are displayed:
start time of treatment
finishing time of treatment
duration of treatment
approximate volume of ultrafiltrate produced

The graph in the middle of the screen displays the pressure inside the circuit.
0 mm Hg represents ambient pressure.
The five small boxes to the right of the pressure display are the pressure limits for the system. New values for these parameters can be entered as required.

Withdrawal Trigger Pressure: this controls the withdrawal of blood from the patient. The system starts at a withdrawal rate of 10 ml/min (30 ml/min if using the fast dialysis system). If the pressure in the system falls below the trigger pressure, the system pauses for 5 seconds and the withdrawal rate is reduced to 9 ml/min before blood withdrawal is restarted. Each time the pressure drops below the trigger pressure the withdrawal rate is reduced by a further 1 ml/min.

With/Ret Limits: these two boxes contain the upper and lower pressure limits in force during the blood withdrawal and blood return phases of the operating cycle. If these limits are exceeded an alarm is triggered and the system is stopped.

Filtration Limits: these limits apply to the filtration phase of the operating cycle. As above, if they are exceeded an alarm is triggered and the system is stopped.

At the bottom left of the screen, the time the system has been running for is displayed. The approximate total volume of filtrate removed in this time is also displayed.

Along the bottom of the screen treatment parameters are displayed. These are self explanatory, apart from:

Weight of Baby: this can be entered during the initialisation procedure, and a calculation is then done to work out the correct blood volume to be withdrawn from the baby. It is not necessary to stick to this calculation - the working blood volume can be entered directly into the Working Blood Volume control.

Alarms: if an alarm is triggered, a red box appears in the middle of the screen and the alarm sounds. The sound can be switched off by pressing the 'OK' button in

the middle of the red box. You must then quit from the user interface, fix the problem and then run the user interface again.

CONSTRUCTION OF BLOOD CIRCUIT

The blood circuit is shown in figure E.2. Connect together the components in the configuration shown. Tap 1 is fitted so that the turning handles point **downwards** as viewed from above. Make sure the top and right hand ports are open and the left hand one closed. Taps 2 and 4 are also fitted with the handles pointing **downwards**, but tap 3 must point **upwards**, and the turning handle positioned so that **all ports are open.**

FLOW RATES

Normal Dialysis System

The blood withdrawal rate starts at 10 ml/min, reducing in increments of 1 ml/min if blood access is inadequate. Blood return is always at 10 ml/min.
The flow rate during the filtration phase is also 10 ml/min.

Fast Dialysis System

The blood withdrawal rate starts at 30 ml/min, again reducing in increments of 1 ml/min.
Blood return is at the same rate as the current withdrawal rate.
The flow rate during filtration is 20 ml/min.

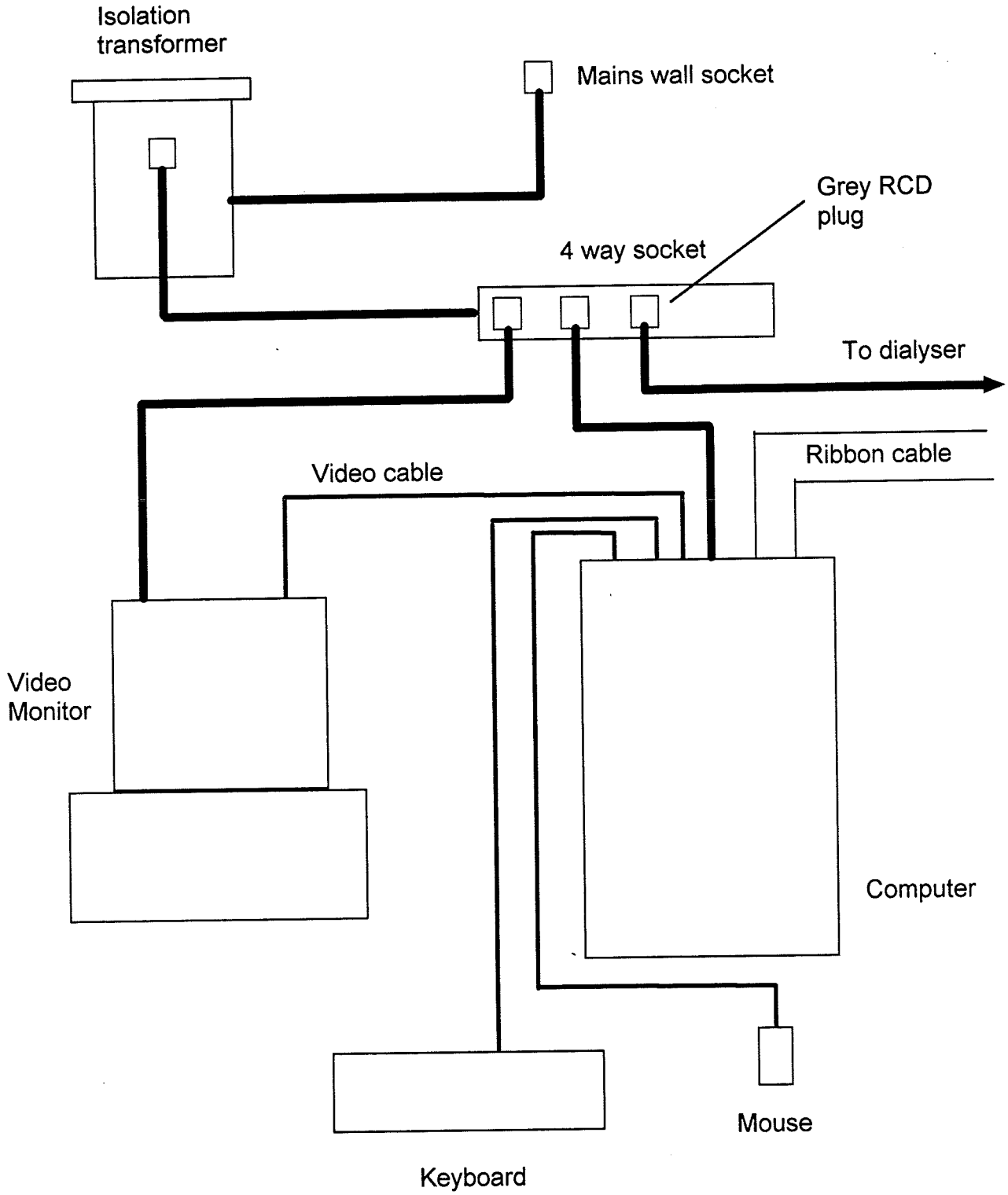


Figure E.1 Connecting the System Together

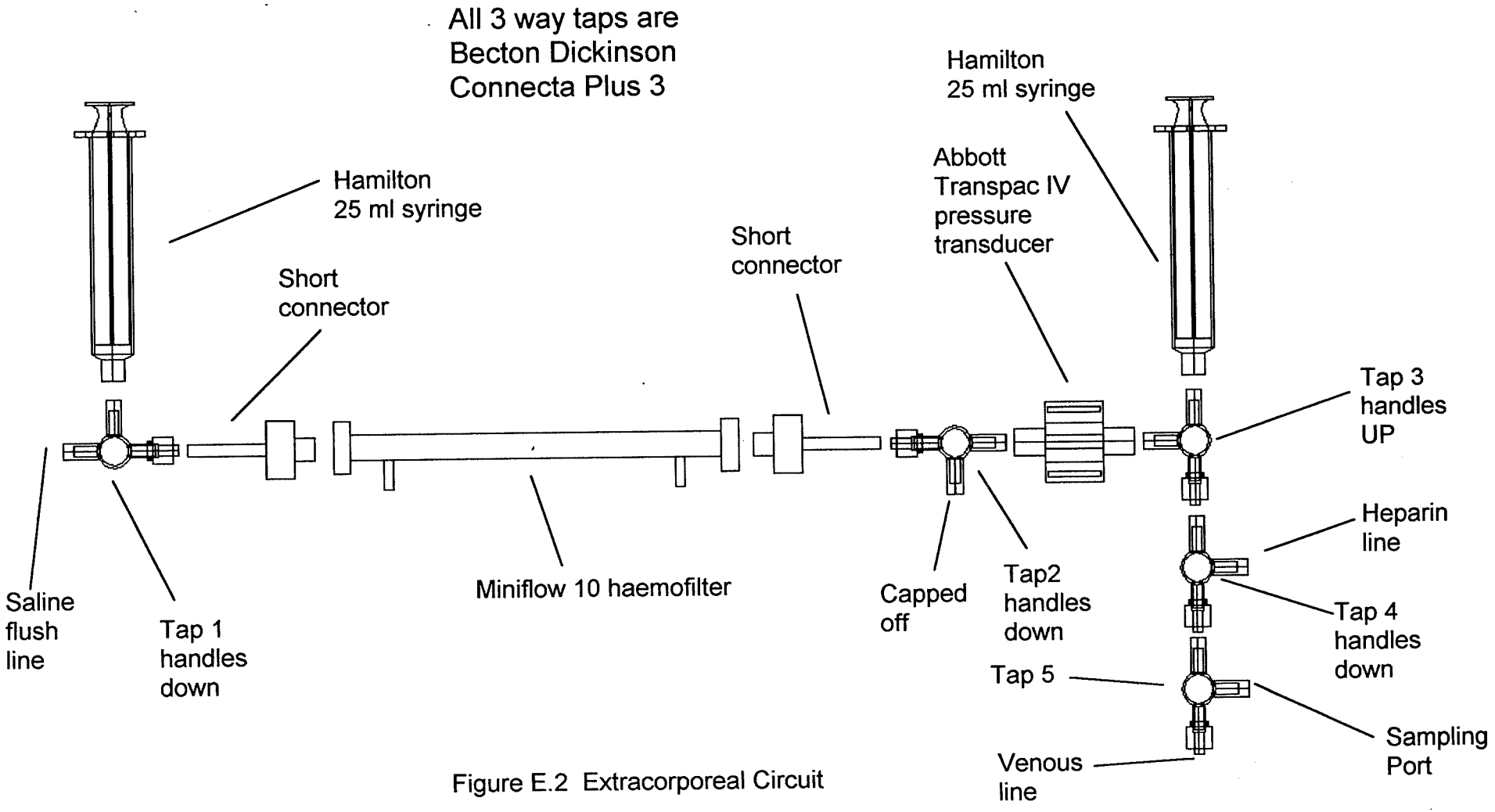


Figure E.2 Extracorporeal Circuit

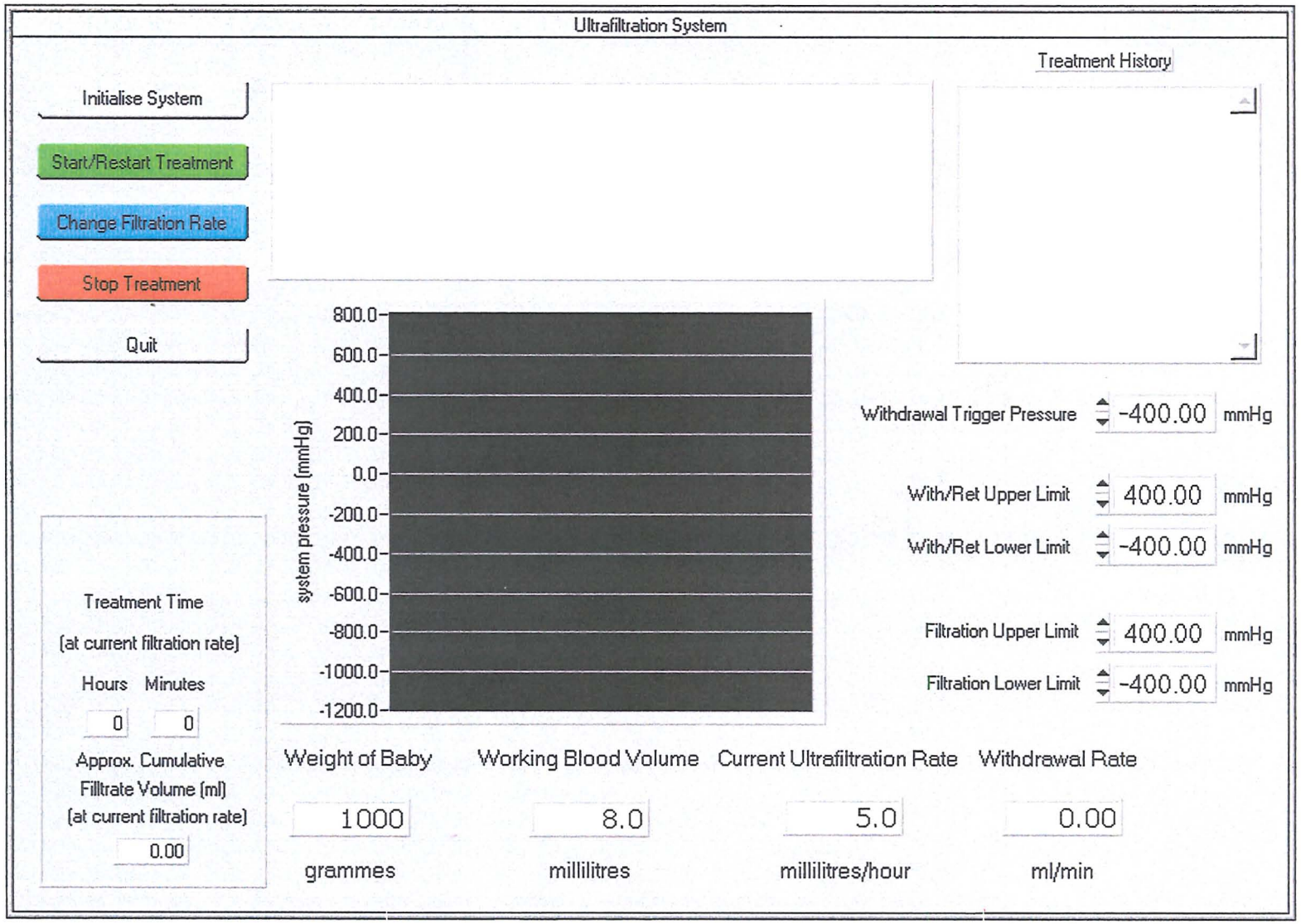


Figure E.3 Graphical User Interface